Glamdring

Automatic Application Partitioning for Intel SGX

Joshua Lind, Christian Priebe, **Divya Muthukumaran**,
Dan O'Keeffe, Pierre-Louis Aublin, Florian Kelbert
Imperial College London

Tobias Reiher
TU Dresden

David Goltzsche
TU Braunschweig

David Eyers
University of Otago

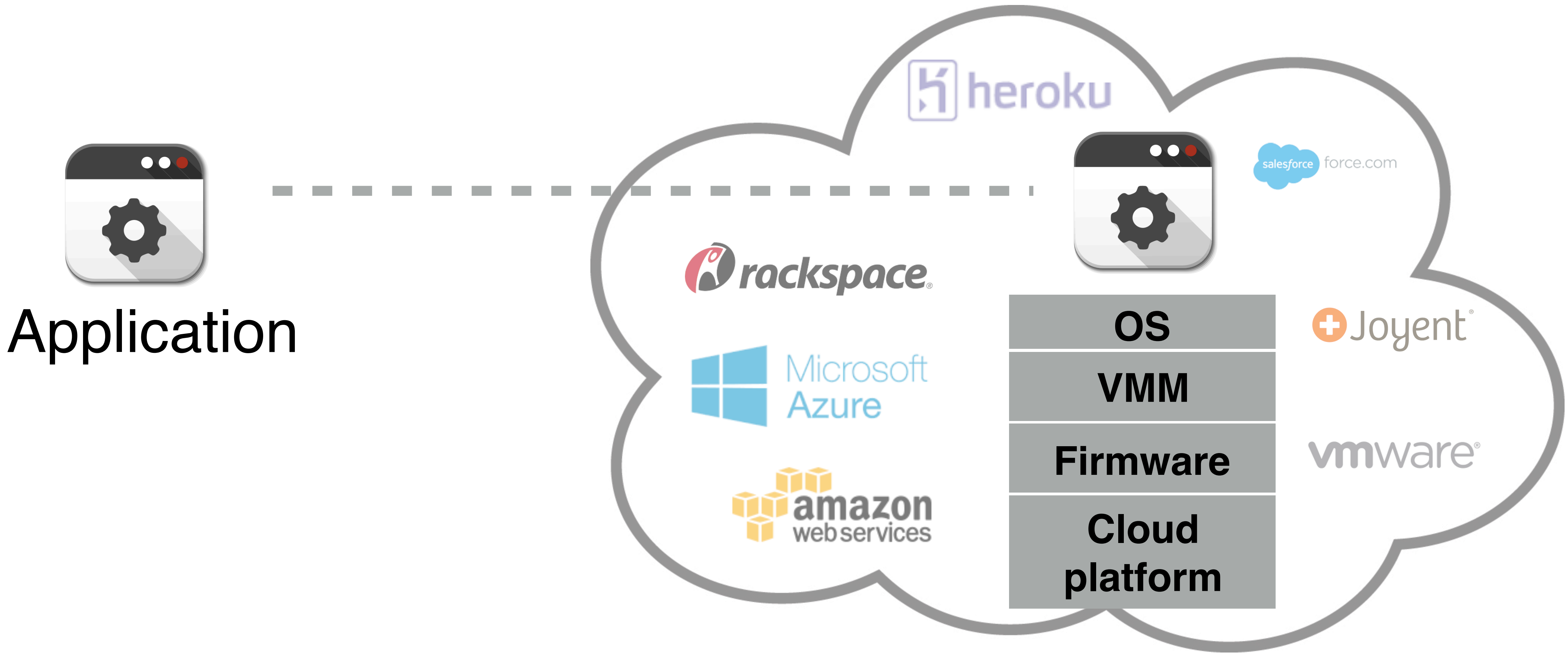
Rüdiger Kapitza
TU Braunschweig

Christof Fetzer
TU Dresden

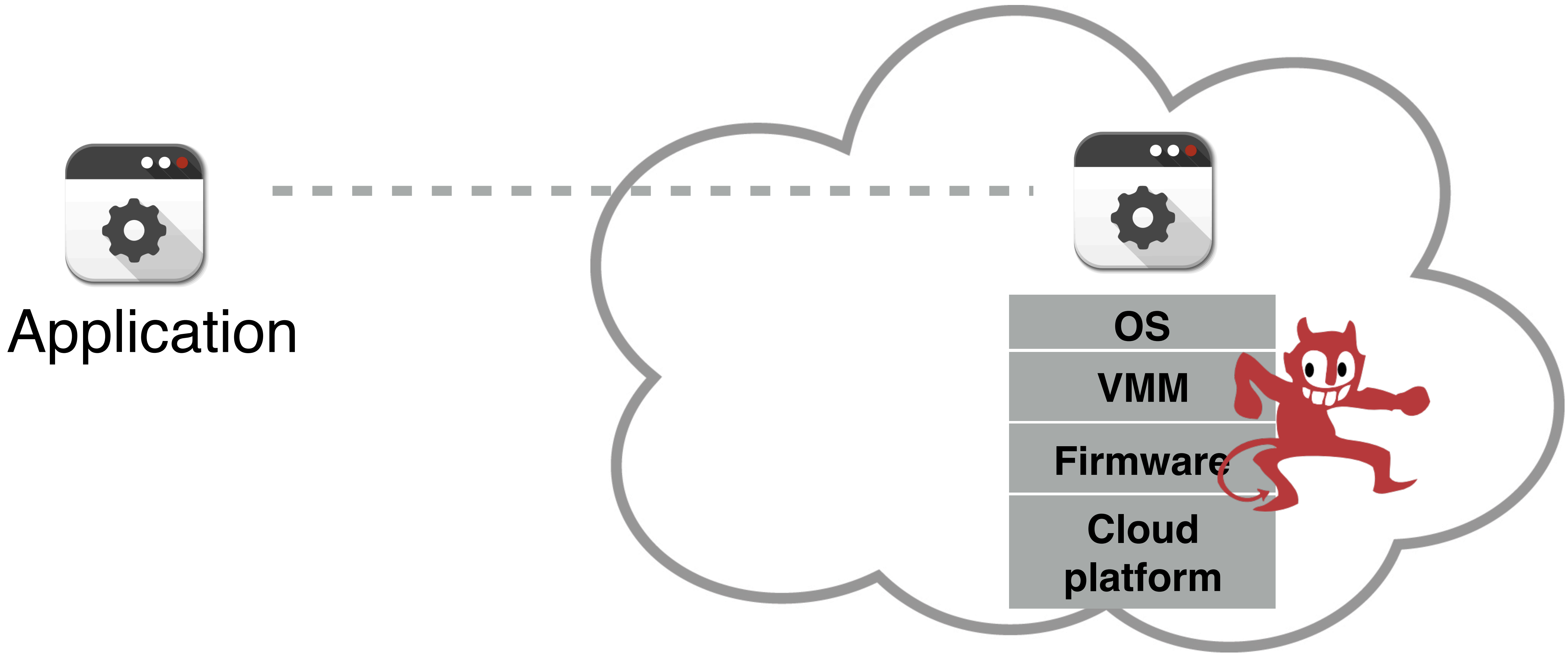
Peter Pietzuch
Imperial College London

dmuthuku@imperial.ac.uk

Trust in Cloud Services



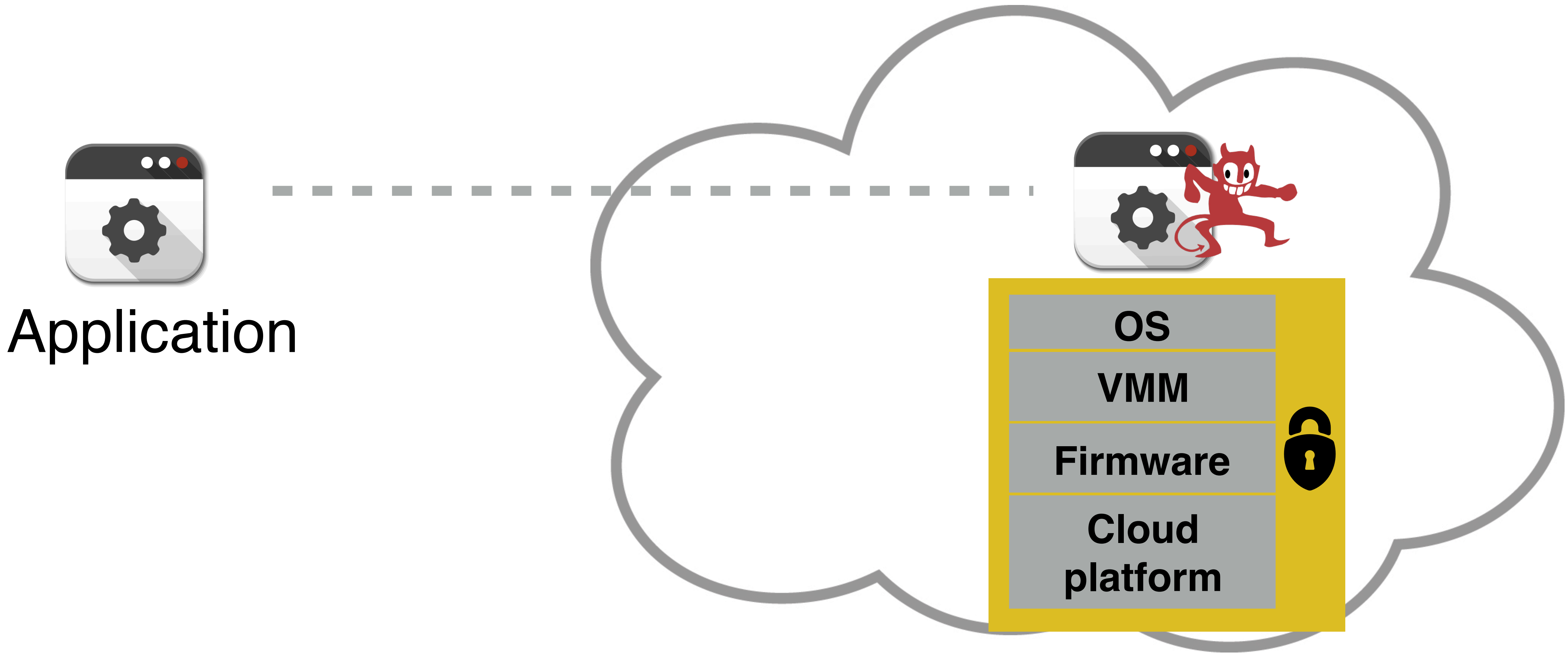
Trust in Cloud Services



Threats

- Insider Attacks
- Human error despite best practices
- Vulnerabilities in large code bases

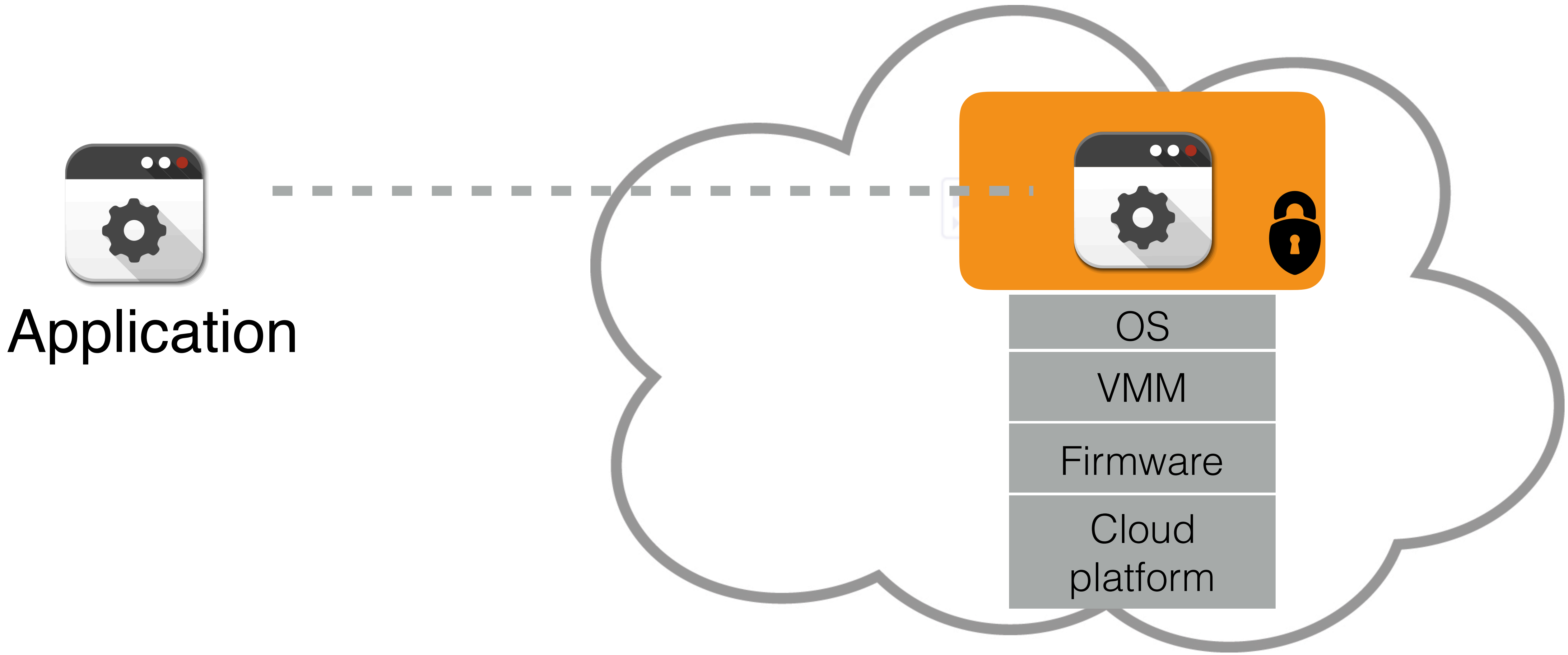
Trust in Cloud Services



Traditional Security Models

- Protect privileged code from untrusted user-level code

Trusted Execution Environments



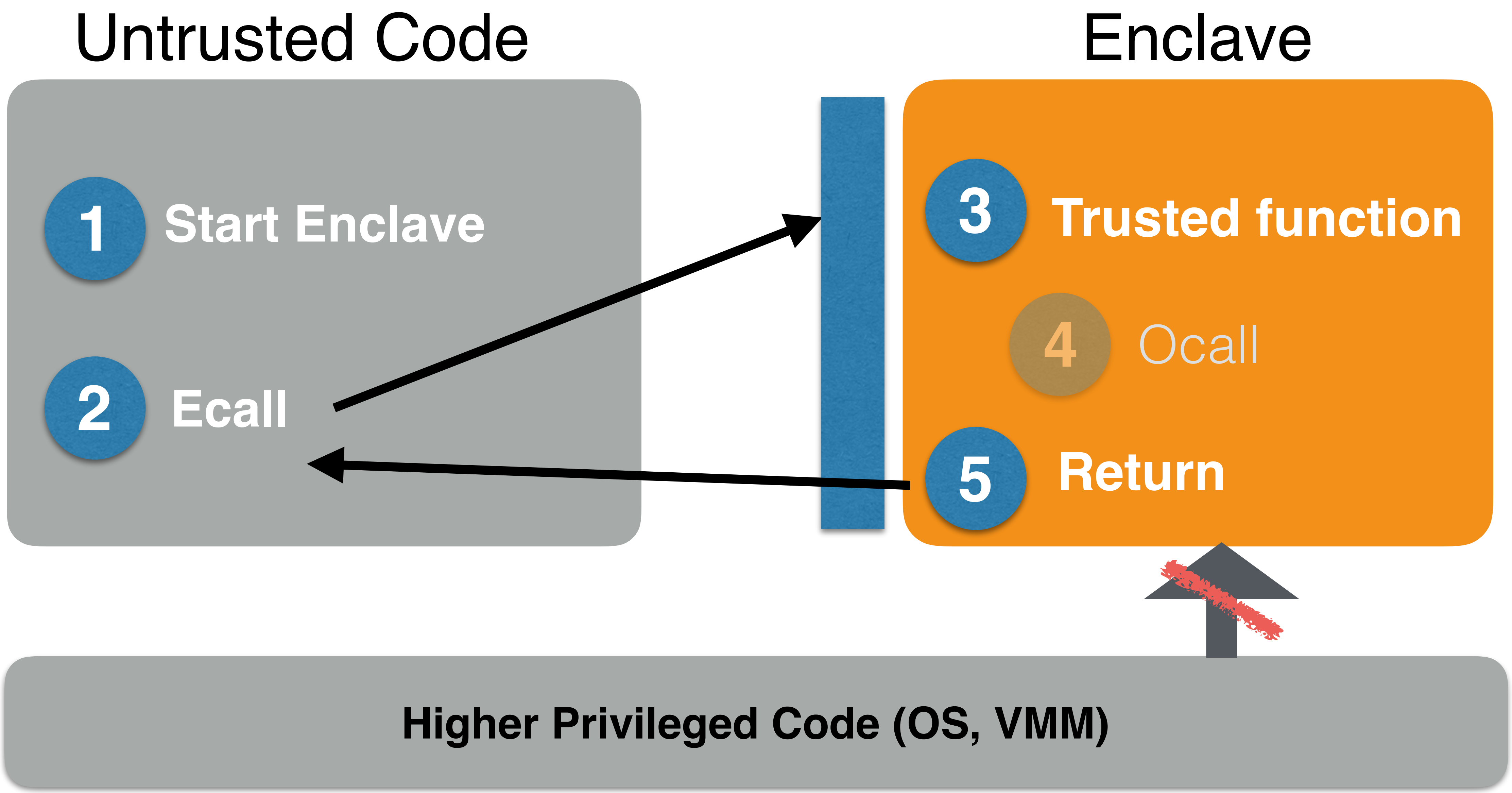
Flips Security Model

- Secure area of a processor
- Provides protection from higher privileged code
- Trusted environment on top of untrusted cloud

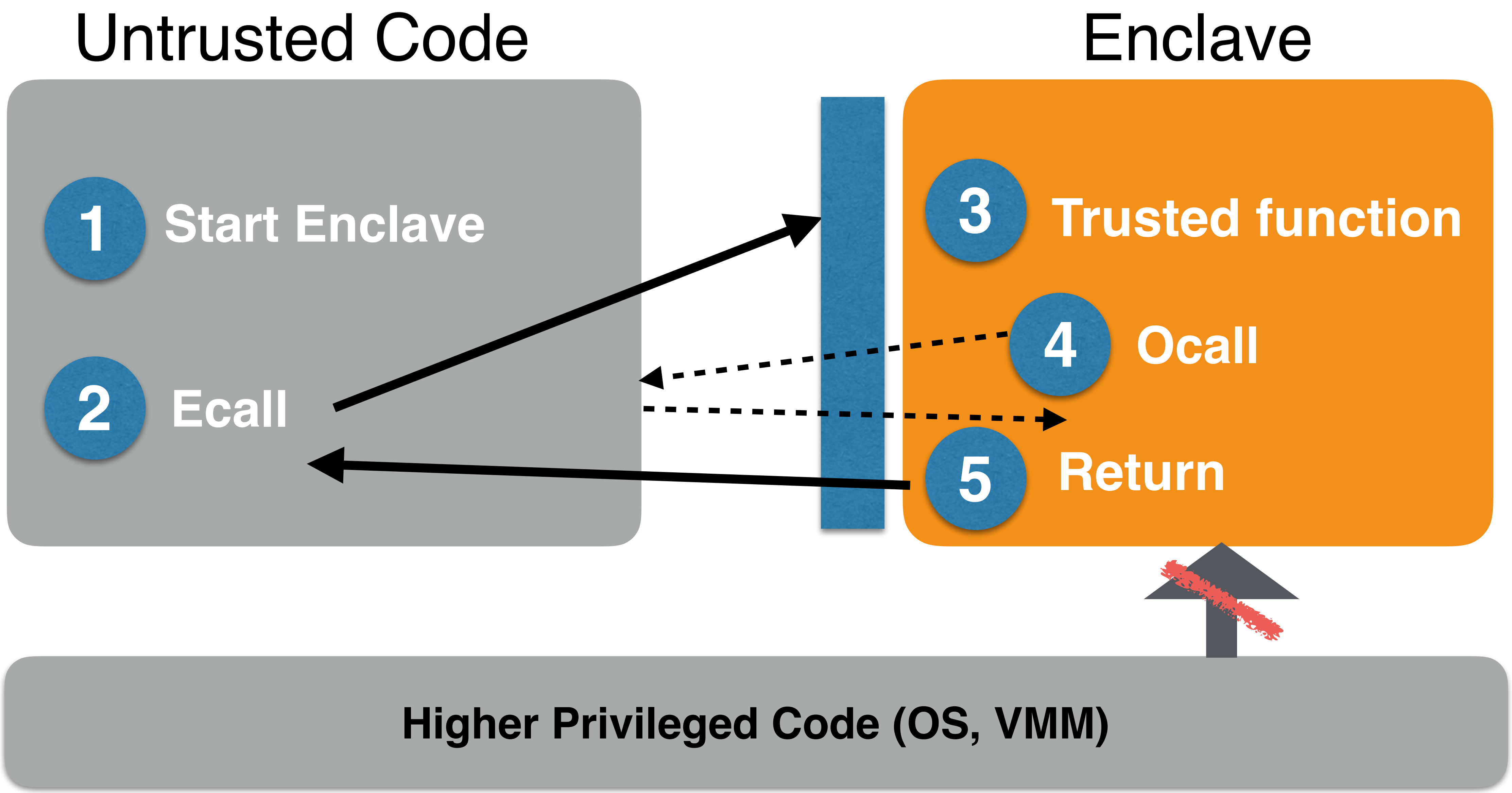
Intel Software Guard Extensions (SGX)

- On commodity processors starting with Skylake
- TEE’s are called enclaves
- 18 CPU instructions to manage enclave lifecycle
- Code & data reside in Enclave Page Cache (EPC)
  - Cache lines encrypted when written to memory
  - Restricted to 128MB
- Intel provides an SDK for Windows and Linux

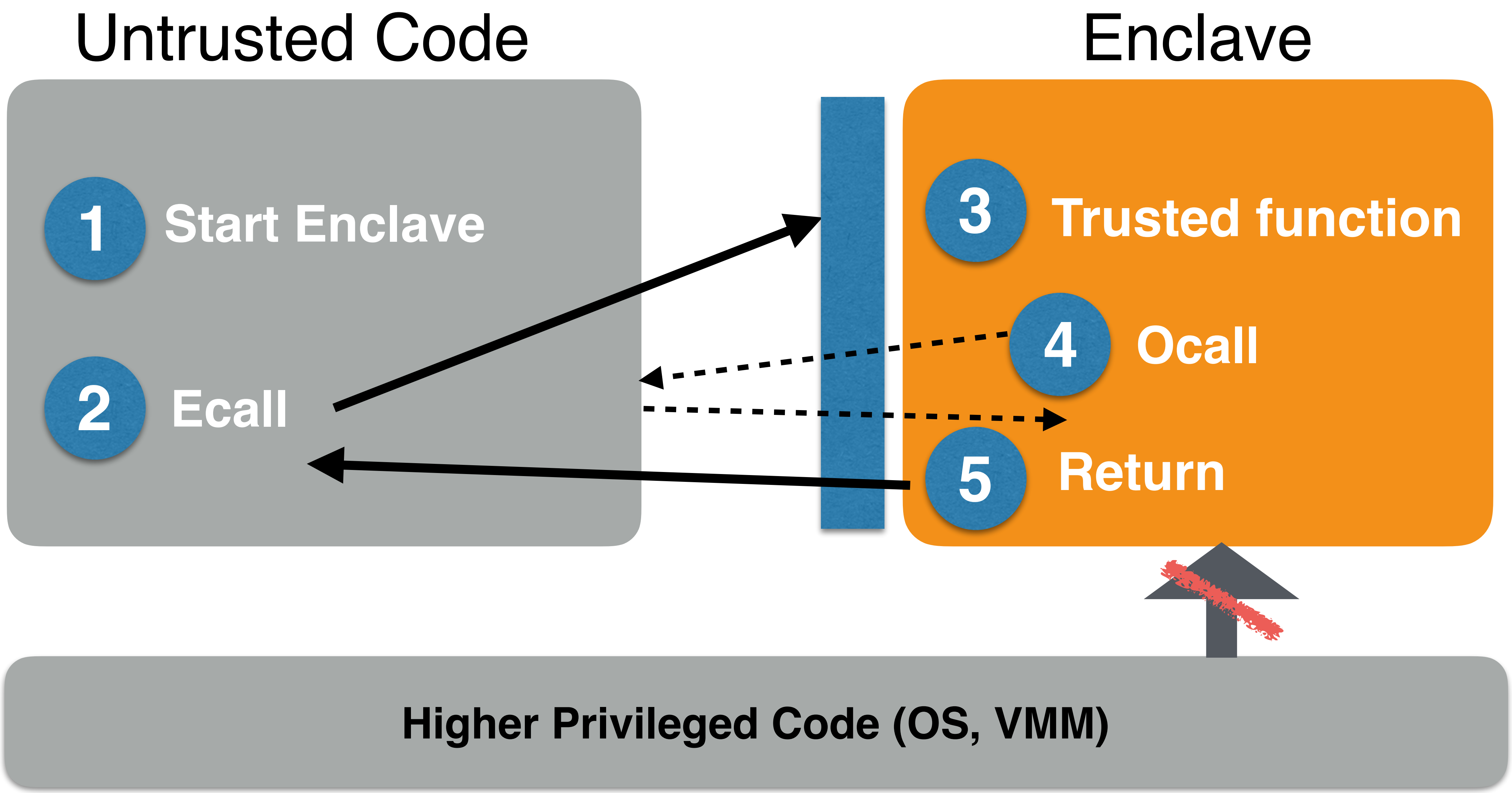
Enclave Application Lifecycle



Enclave Application Lifecycle

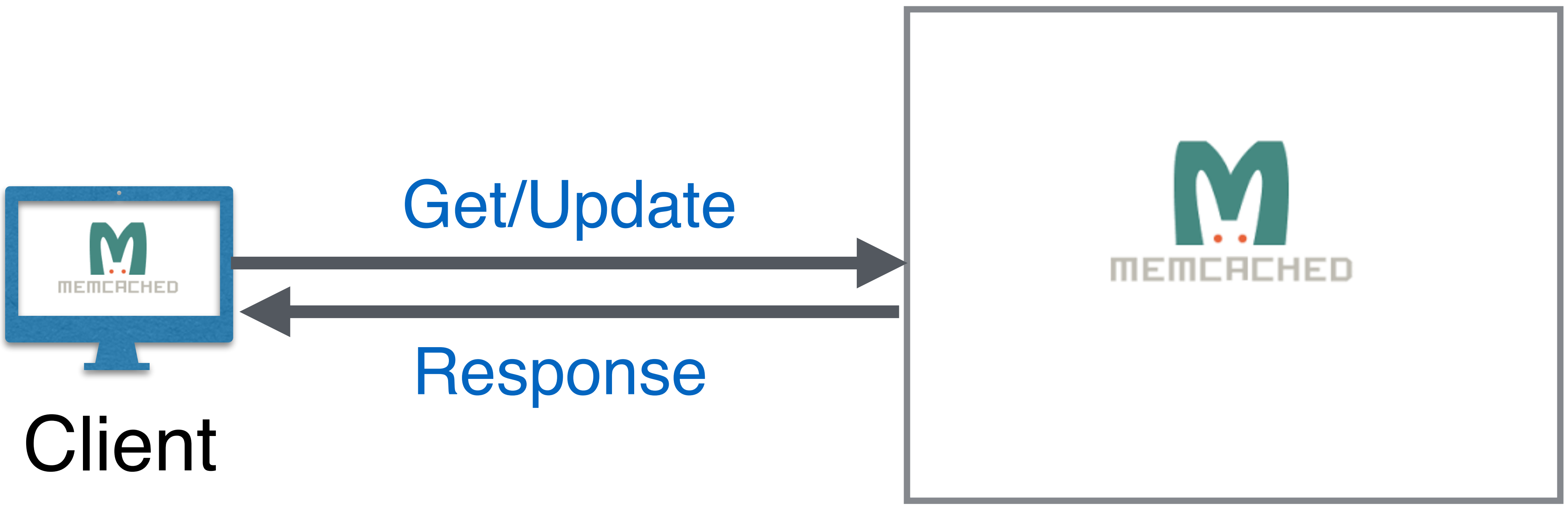


Enclave Application Lifecycle



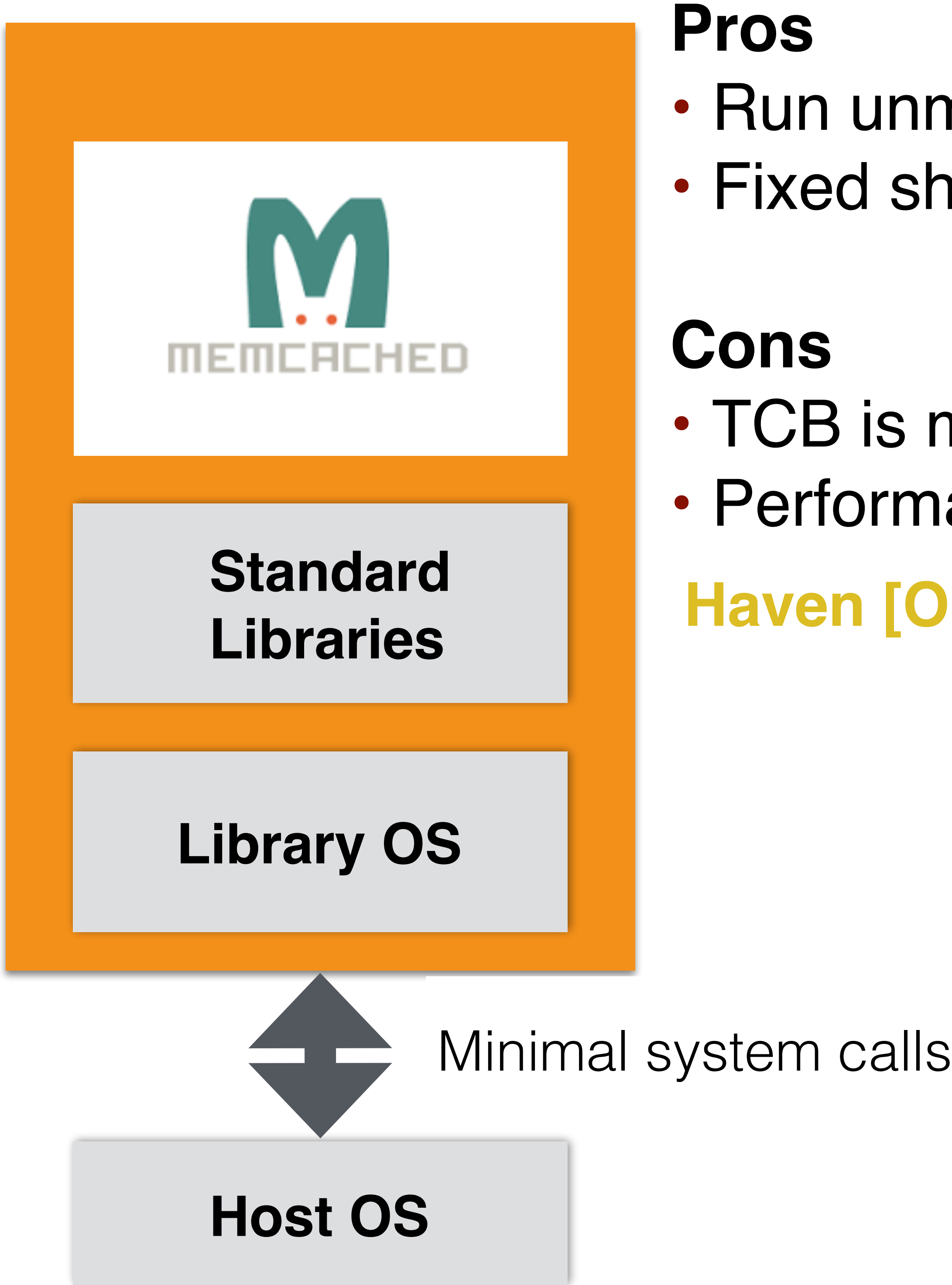
Enclave crossings through ecalls and ocalls incur a performance penalty

Porting applications to Enclaves



How do you port a key-value store to run in an enclave?

Library OS Inside Enclaves

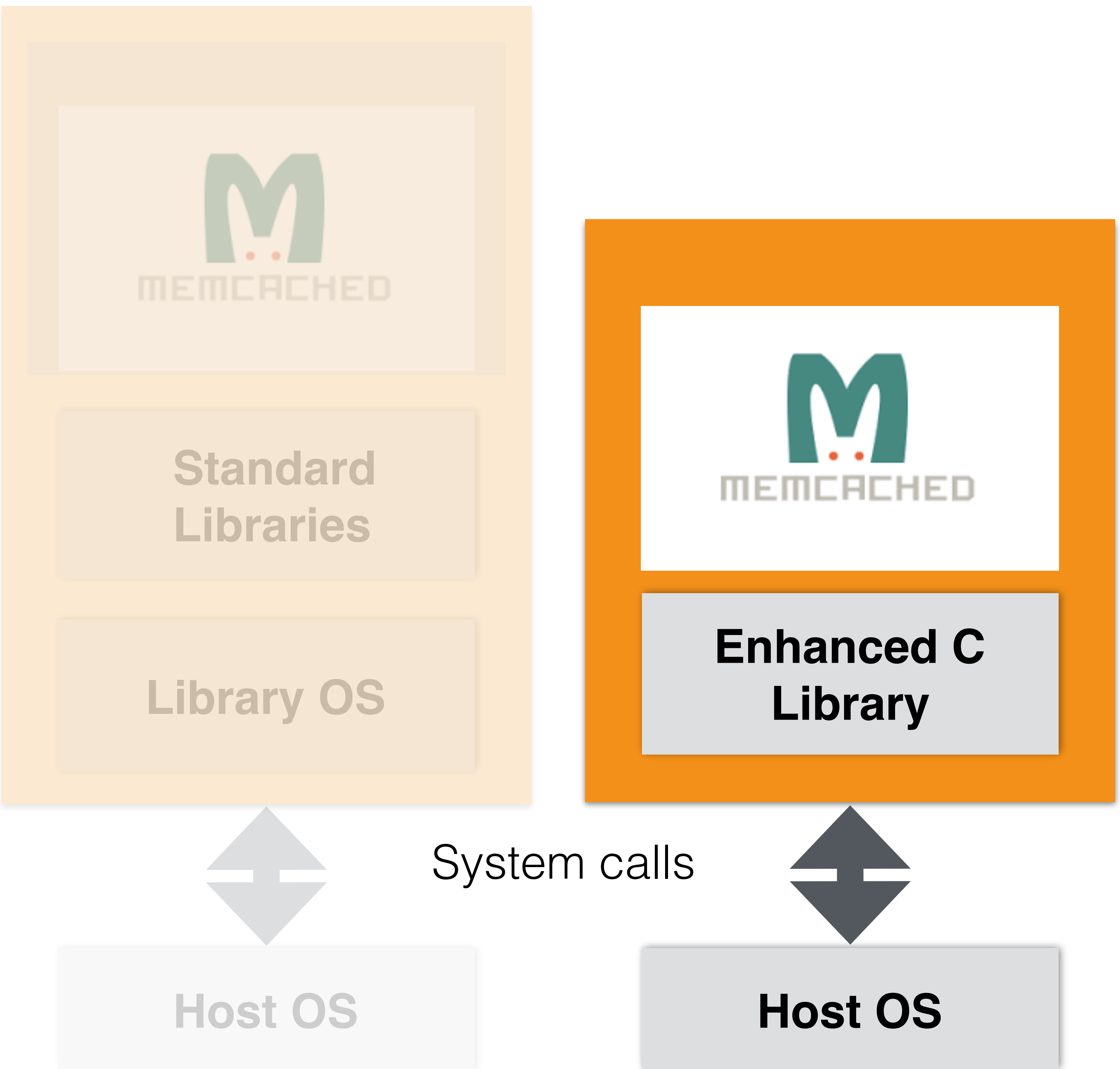


Pros
- Run unmodified applications
- Fixed shielded interface

Cons
- TCB is millions LoC!
- Performance overhead

Haven [OSDI'14]

Standard Library Inside Enclaves



Pros

- Smaller TCB than Haven
- Fixed shielded interface

Cons

- TCB = 0.6x–2x of application size
- Recompilation needed

SCONE [OSDI'16]

Minimum TCB Inside Enclaves

Principle of Least Privilege

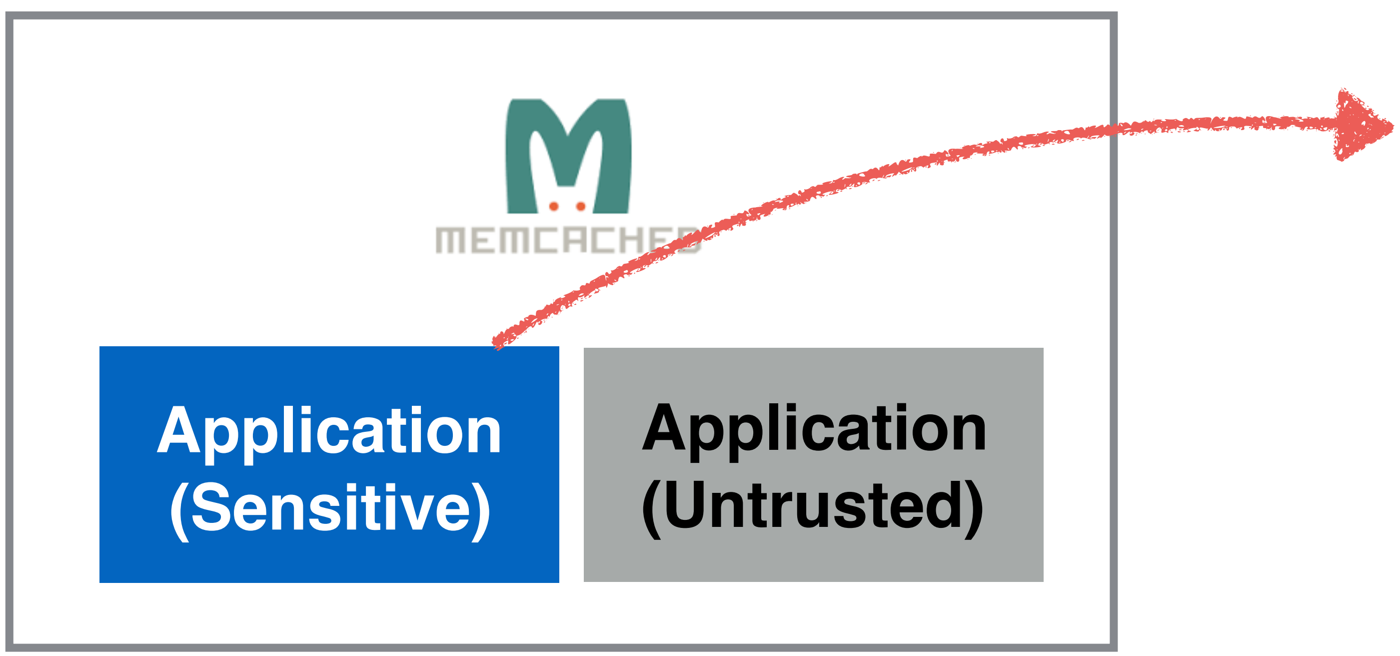
Only move the code needed to enforce security policy

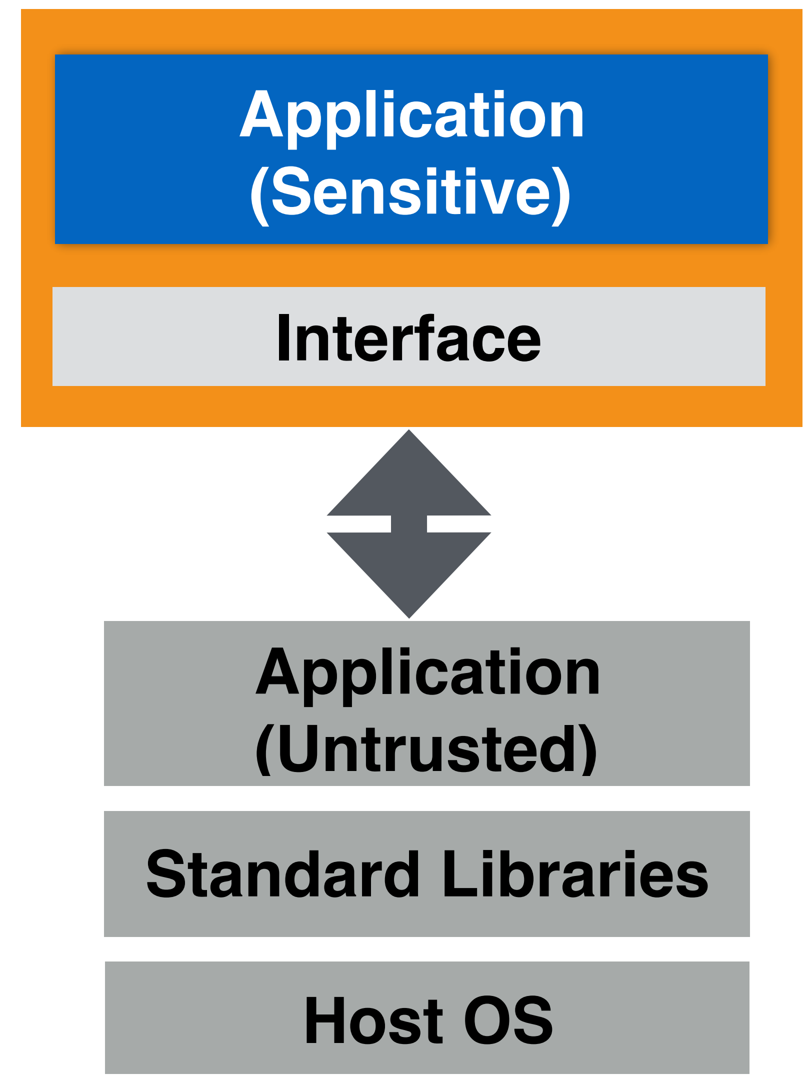
Policy: Confidentiality and Integrity of key-value pairs

Minimum TCB Inside Enclaves

Principle of Least Privilege

Only move the code needed to enforce security policy





Policy: Confidentiality and Integrity of key-value pairs

Application Partitioning to Minimise TCB

Prior work has **manually** partitioned applications

SecureKeeper: Confidential ZooKeeper using Intel SGX

Stefan Brenner
TU Braunschweig, Germany
brenner@ibr.cs.tu-bs.de

Colin Wulf
TU Braunschweig, Germany
cwulf@ibr.cs.tu-bs.de

David Goltzsche
TU Braunschweig, Germany
goltzsche@ibr.cs.tu-bs.de

Nico Weichbrodt
TU Braunschweig, Germany
weichbr@ibr.cs.tu-bs.de

Matthias Lorenz
TU Braunschweig, Germany
mlorenz@ibr.cs.tu-bs.de

Christof Fetzer
TU Dresden, Germany
christof.fetzer@tu-dresden.de

Peter Pietzuch
Imperial College London, UK
prp@imperial.ac.uk

Rüdiger Kapitza
TU Braunschweig, Germany
rrkap

ABSTRACT

Cloud computing, while ubiquitous, still suffers from trust issues, especially for applications managing sensitive data.

1. IN

Cloud
fits to b

2015 IEEE Symposium on Security and Privacy

VC3: Trustworthy Data Analytics in the Cloud using SGX

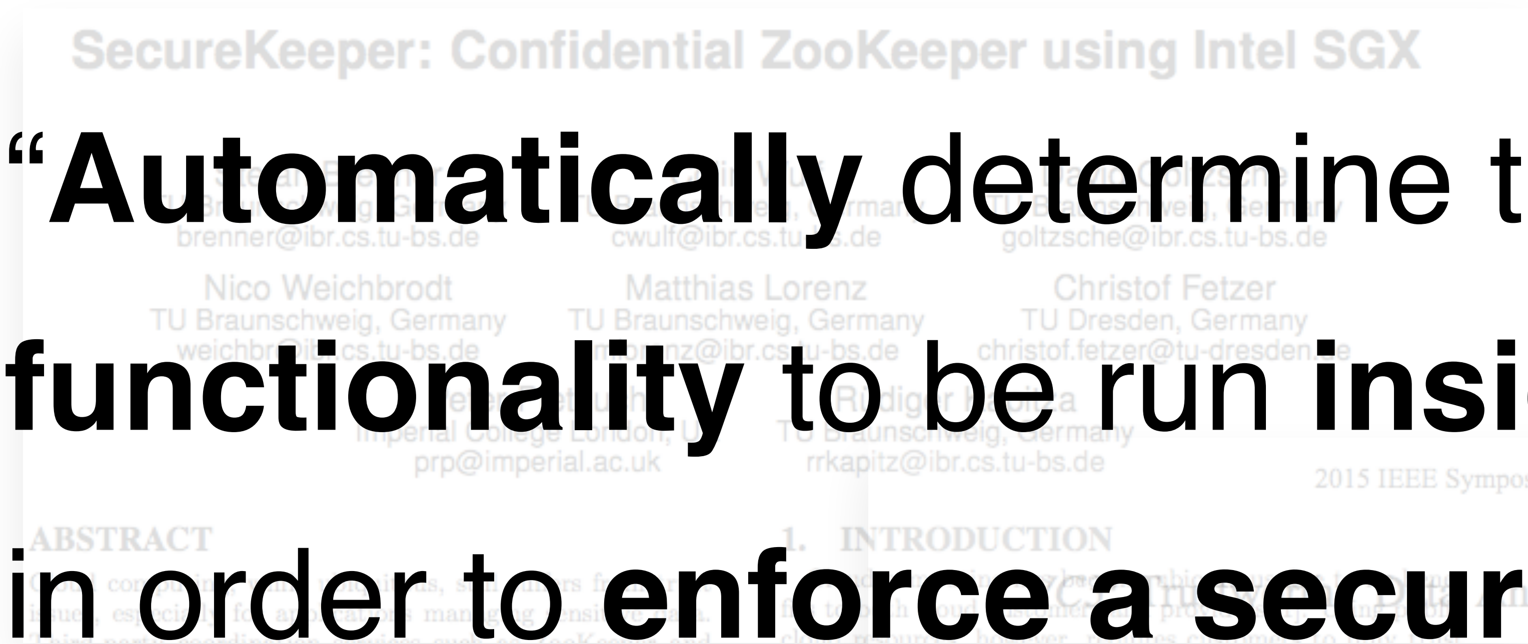
Felix Schuster*, Manuel Costa, Cédric Fournet, Christos Gkantsidis
Marcus Peinado, Gloria Mainar-Ruiz, Mark Russinovich
Microsoft Research

***Abstract*—We present *VC3*, the first system that allows users to run distributed MapReduce computations in the cloud while keeping their code and data secret, and ensuring the correctness and completeness of their results. *VC3* runs on unmodified Hadoop, but crucially keeps Hadoop, the operating system and the hypervisor out of the TCB; thus, confidentiality and integrity**

data [22]. However, FHE is not efficient for most computations [23], [65]. The computation can also be shared between independent parties while guaranteeing confidentiality for individual inputs (using e. g., garbled circuits [29]) and providing protection against corrupted parties (see e. g.,

Application Partitioning to Minimise TCB

Prior work has **manually** partitioned applications


SecureKeeper: Confidential ZooKeeper using Intel SGX

brenner@ibr.cs.tu-bs.de

goltzsche@ibr.cs.tu-bs.de

Nico Weichbrodt
TU Braunschweig, Germany

Matthias Lorenz
TU Braunschweig, Germany

Christof Fetzer
TU Dresden, Germany

prp@imperial.ac.uk

rrkapitz@ibr.cs.tu-bs.de

ABSTRACT

1. INTRODUCTION


2015 IEEE Symposium on Security and Privacy

Felix Schuster*, Manuel Costa, Cédric Fournet, Christos Gkantsidis
Marcus Peinado, Gloria Mainar-Ruiz, Mark Russinovich
Microsoft Research

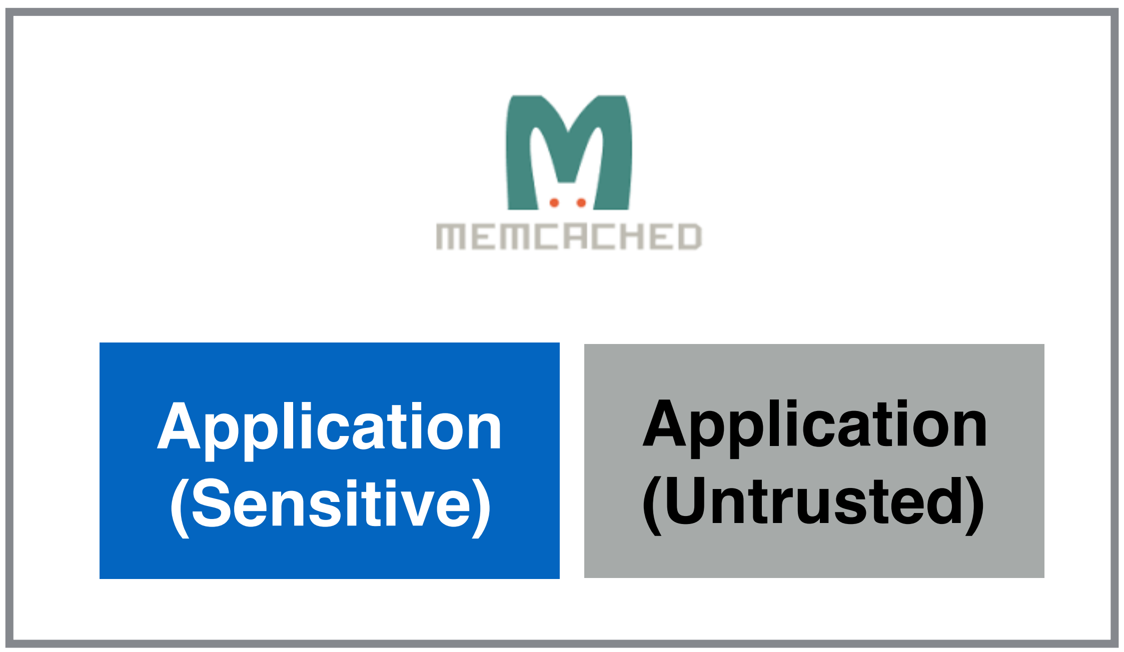
Abstract—We present *VC3*, the first system that allows users to run distributed MapReduce computations in the cloud while keeping their code and data secret, and ensuring the correctness and completeness of their results. *VC3* runs on unmodified Hadoop, but crucially keeps Hadoop, the operating system and the hypervisor out of the TCB; thus, confidentiality and integrity

data [22]. However, FHE is not efficient for most computations [23], [65]. The computation can also be shared between independent parties while guaranteeing confidentiality for individual inputs (using e.g., garbled circuits [29]) and providing protection against corrupted parties (see e.g.,

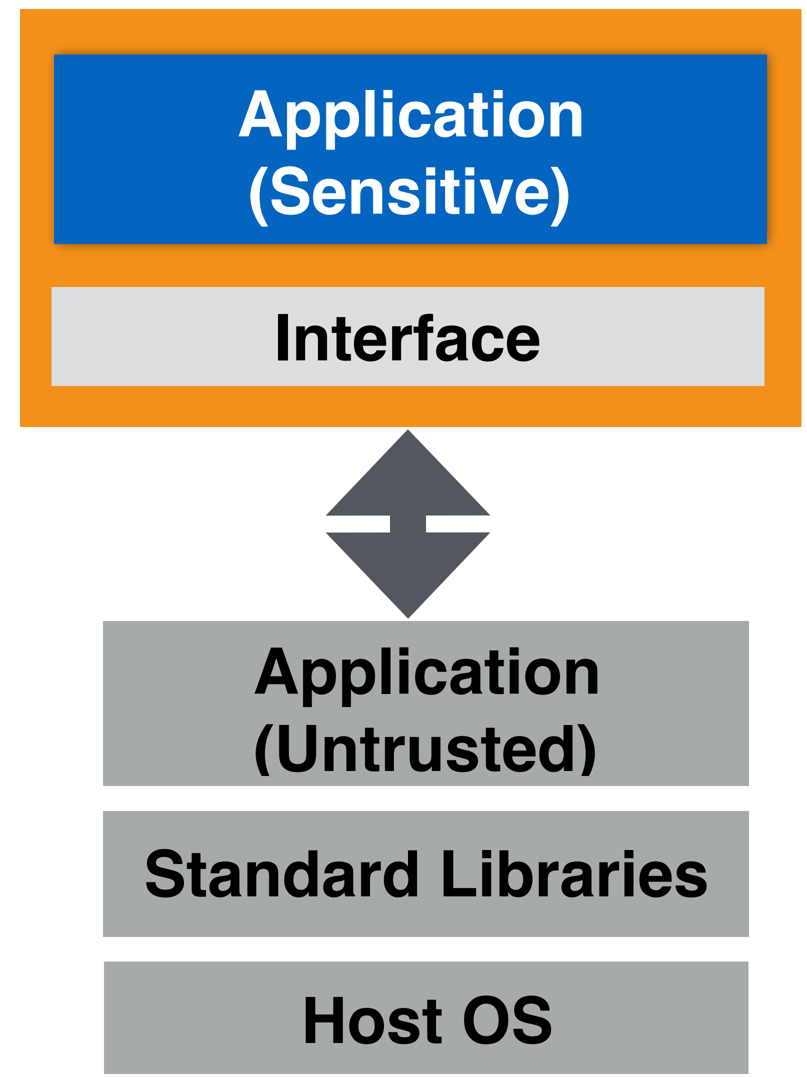
“**Automatically** determine the **minimum functionality** to be run **inside an enclave** in order to **enforce a security policy**”

Challenges in Automated Partitioning

- Identifying security-sensitive code relevant to a security policy
- Preventing interfaces from violating security policy
- Avoiding performance degradation

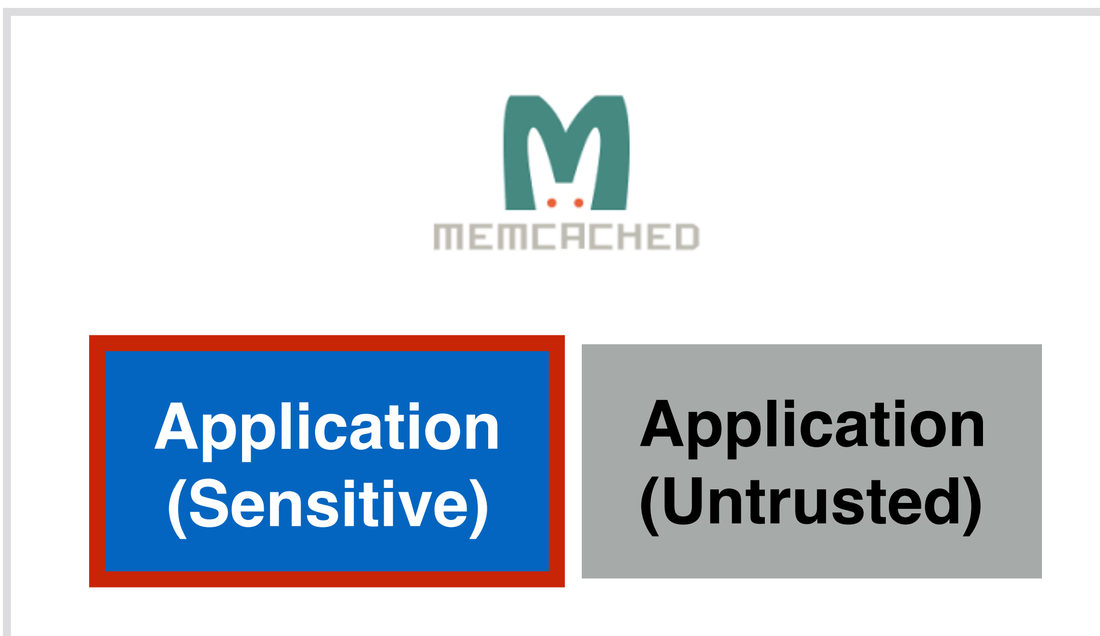


Policy: Confidentiality and Integrity of key-value pairs

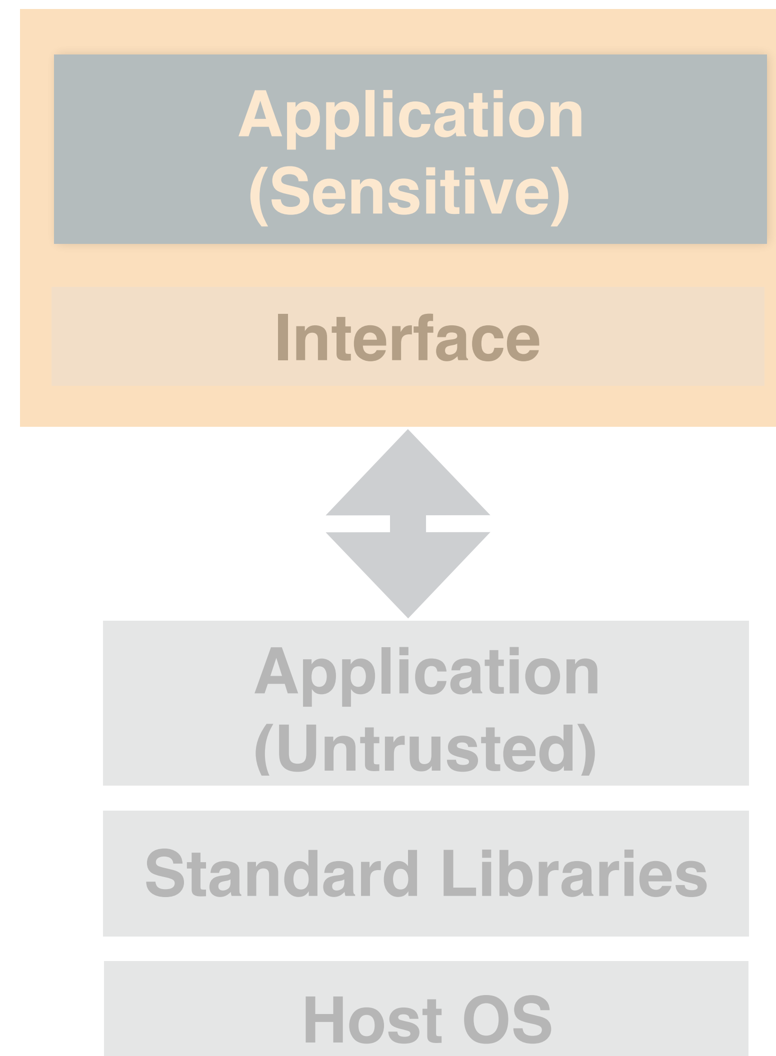


Challenges in Automated Partitioning

- Identifying security-sensitive code relevant to a security policy
- Preventing interfaces from violating security policy
- Avoiding performance degradation

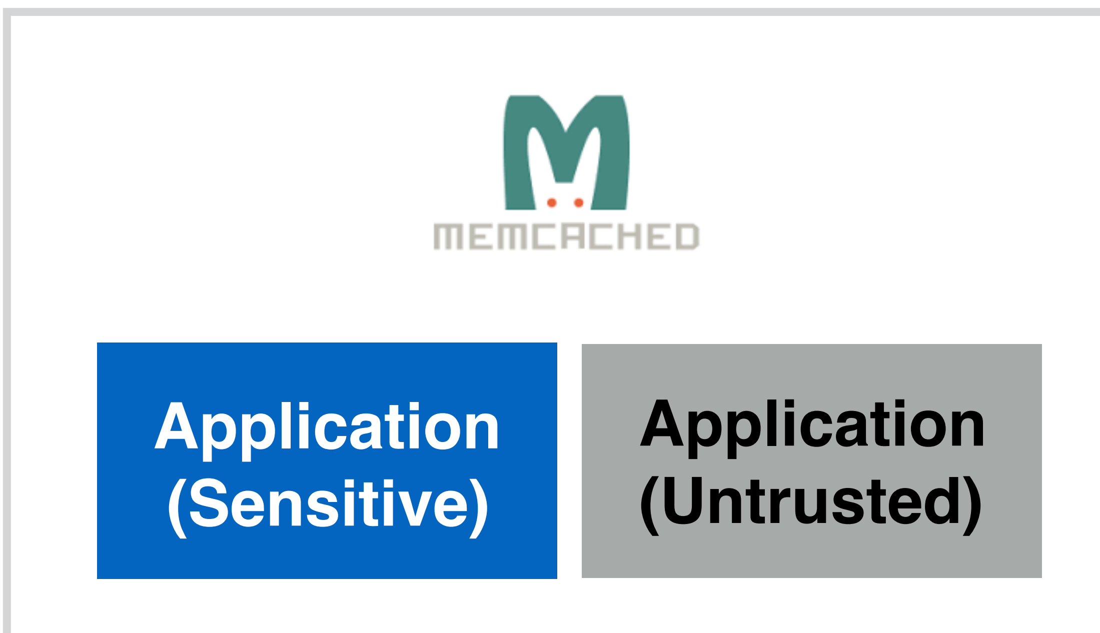


Policy: Confidentiality and Integrity of key-value pairs

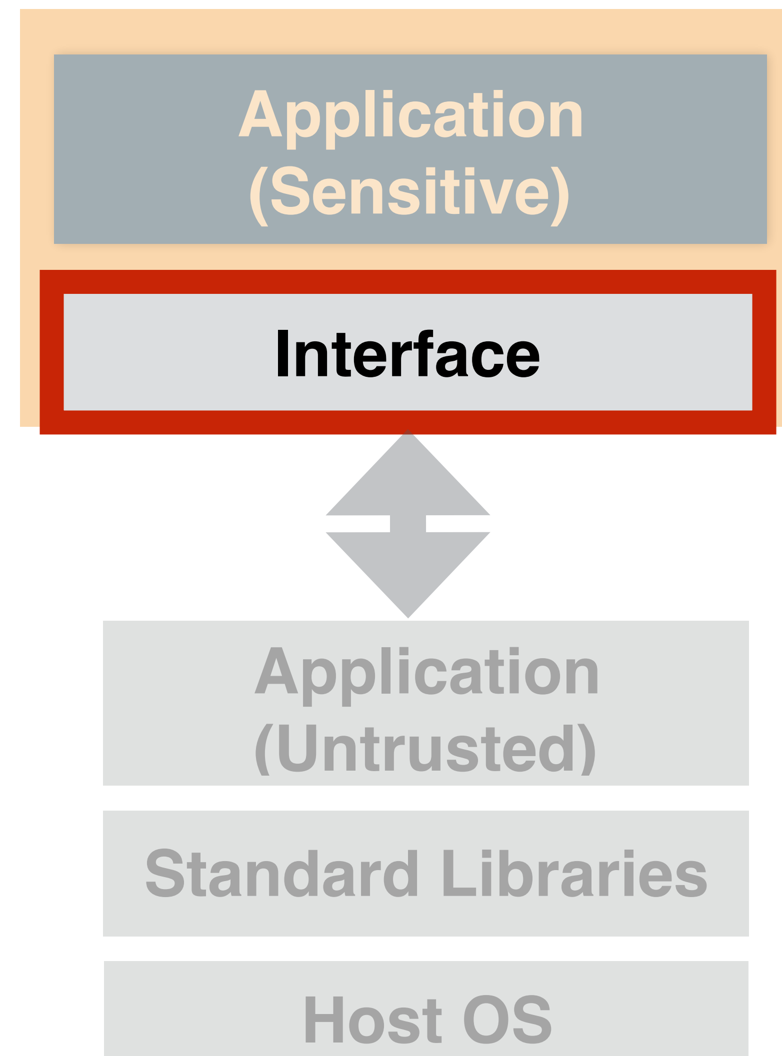


Challenges in Automated Partitioning

- Identifying security-sensitive code relevant to a security policy
- Preventing interfaces from violating security policy
- Avoiding performance degradation

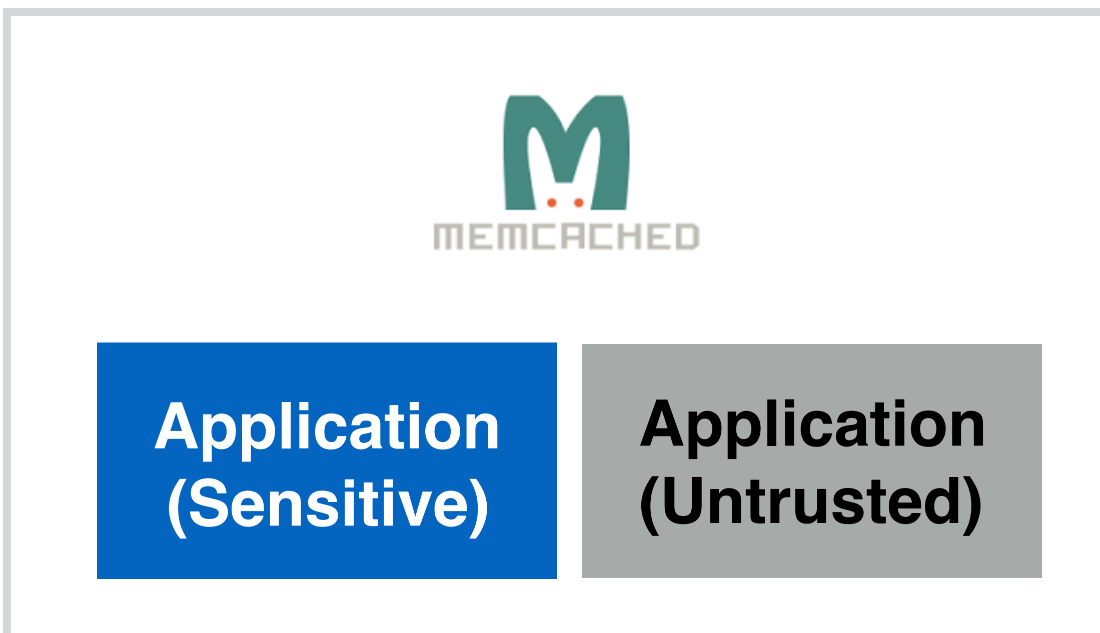


Policy: Confidentiality and Integrity of key-value pairs

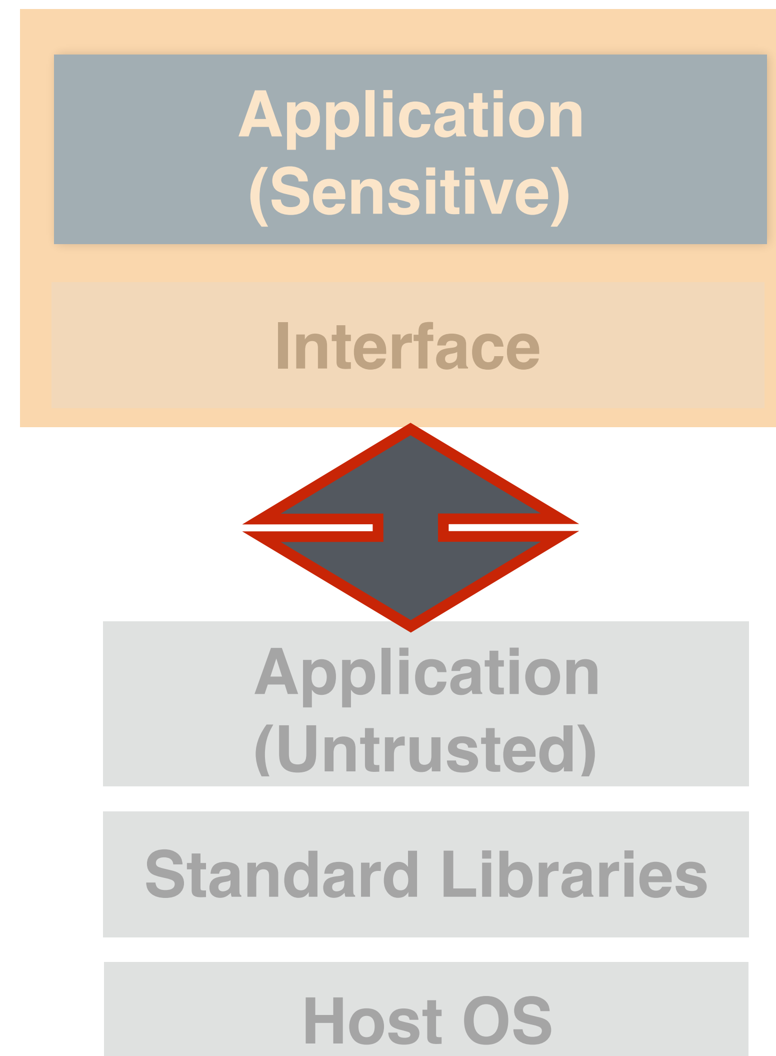


Challenges in Automated Partitioning

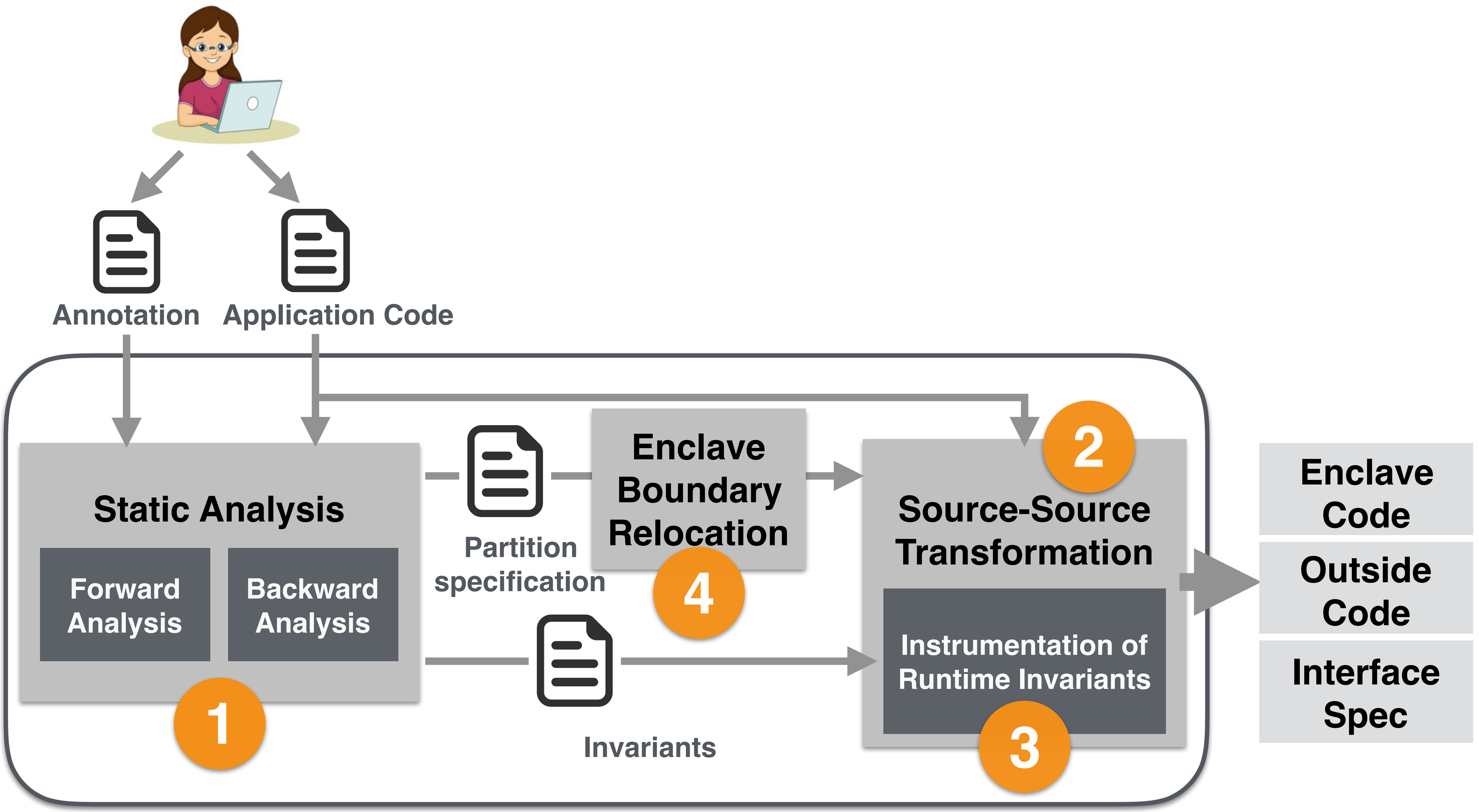
- Identifying security-sensitive code relevant to a security policy
- Preventing interfaces from violating security policy
- Avoiding performance degradation



Policy: Confidentiality and Integrity of key-value pairs

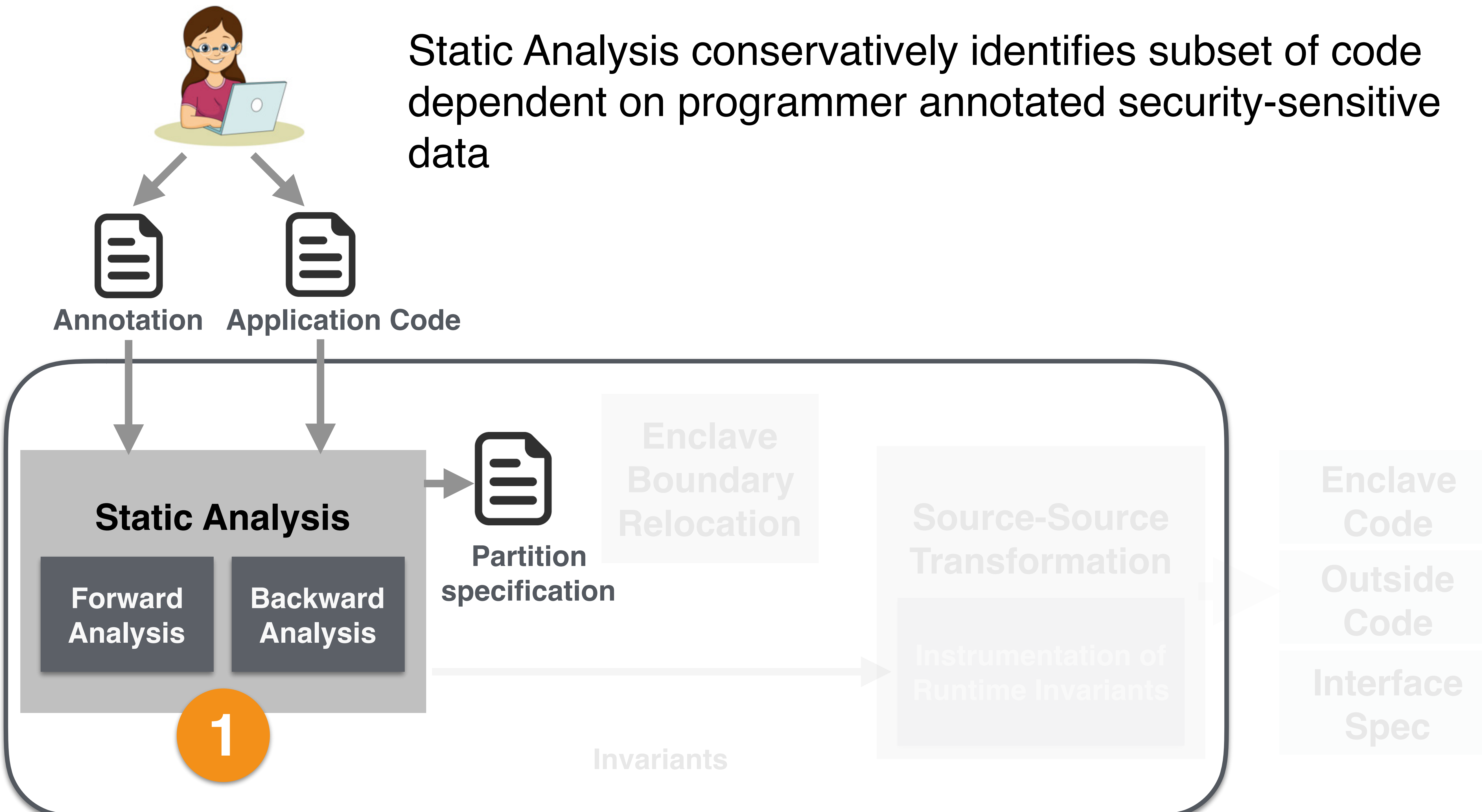


Glamdring Partitioning Framework

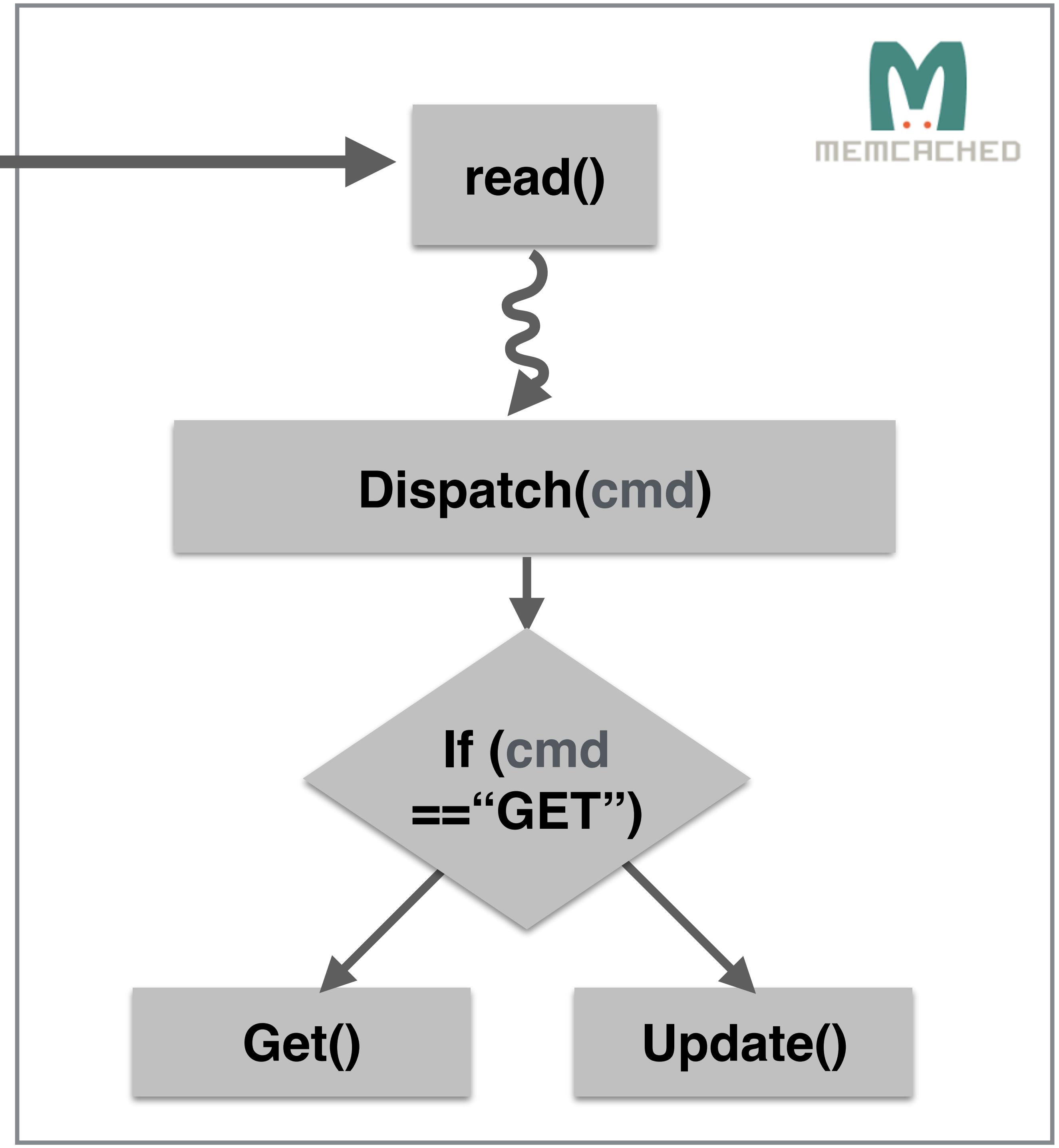


1. Identify Security-Sensitive Code

Static Analysis conservatively identifies subset of code dependent on programmer annotated security-sensitive data



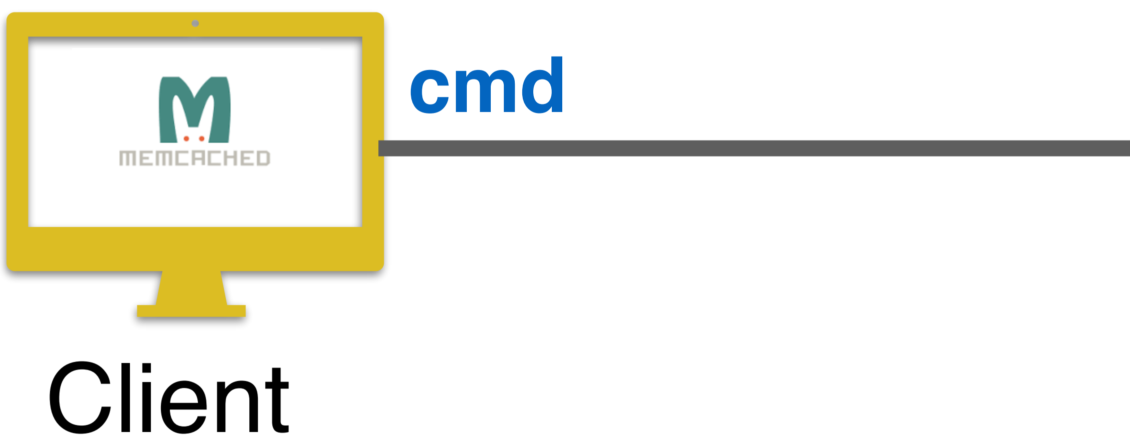
Annotation of Security-Sensitive Data

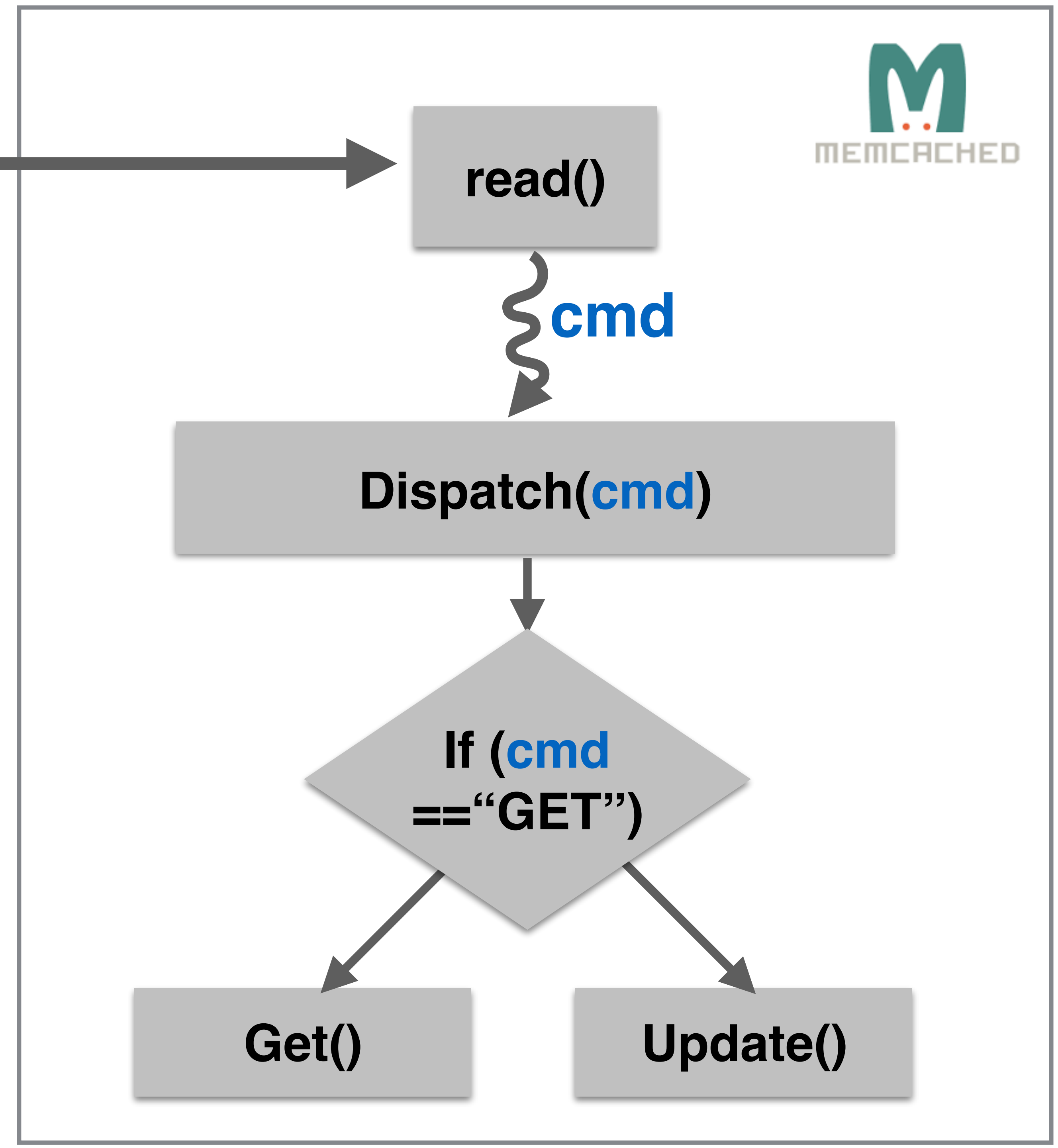


What to Annotate

- Indicate where security-sensitive data enters or leaves the program
- Security-sensitive data can be **encrypted** and **signed** until first use

Annotation of Security-Sensitive Data

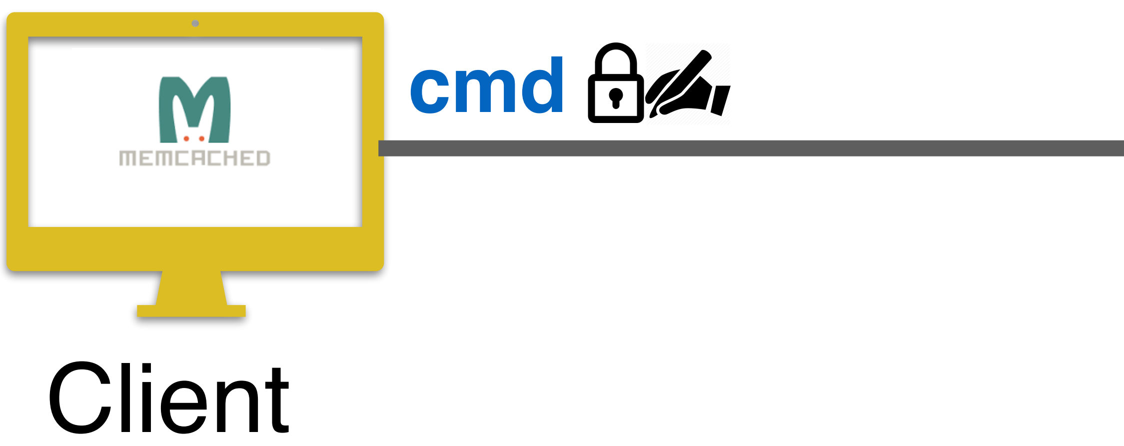


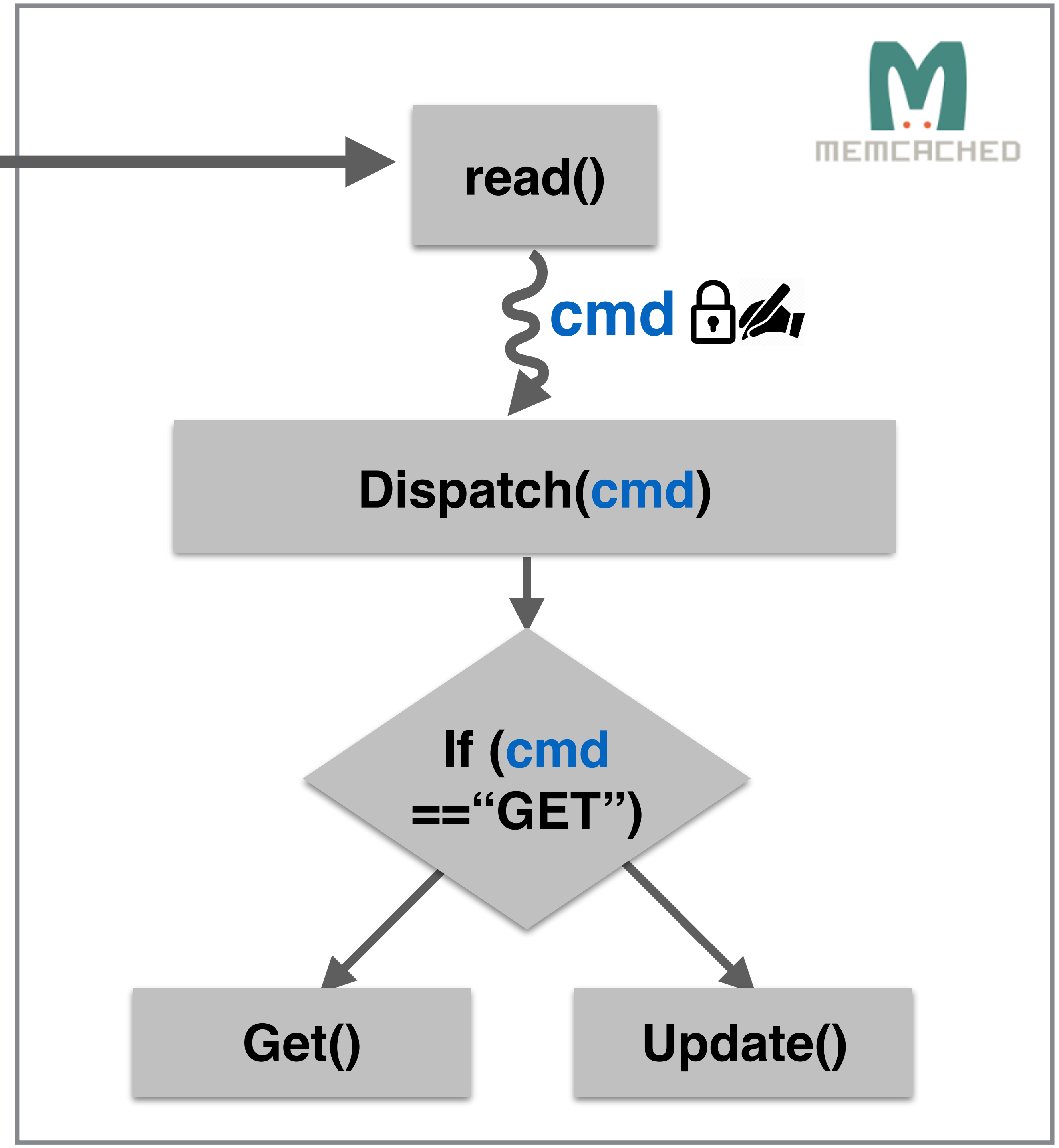


What to Annotate

- Indicate where security-sensitive data enters or leaves the program
- Sensitive data can be **encrypted** and **signed** until first use

Annotation of Security-Sensitive Data

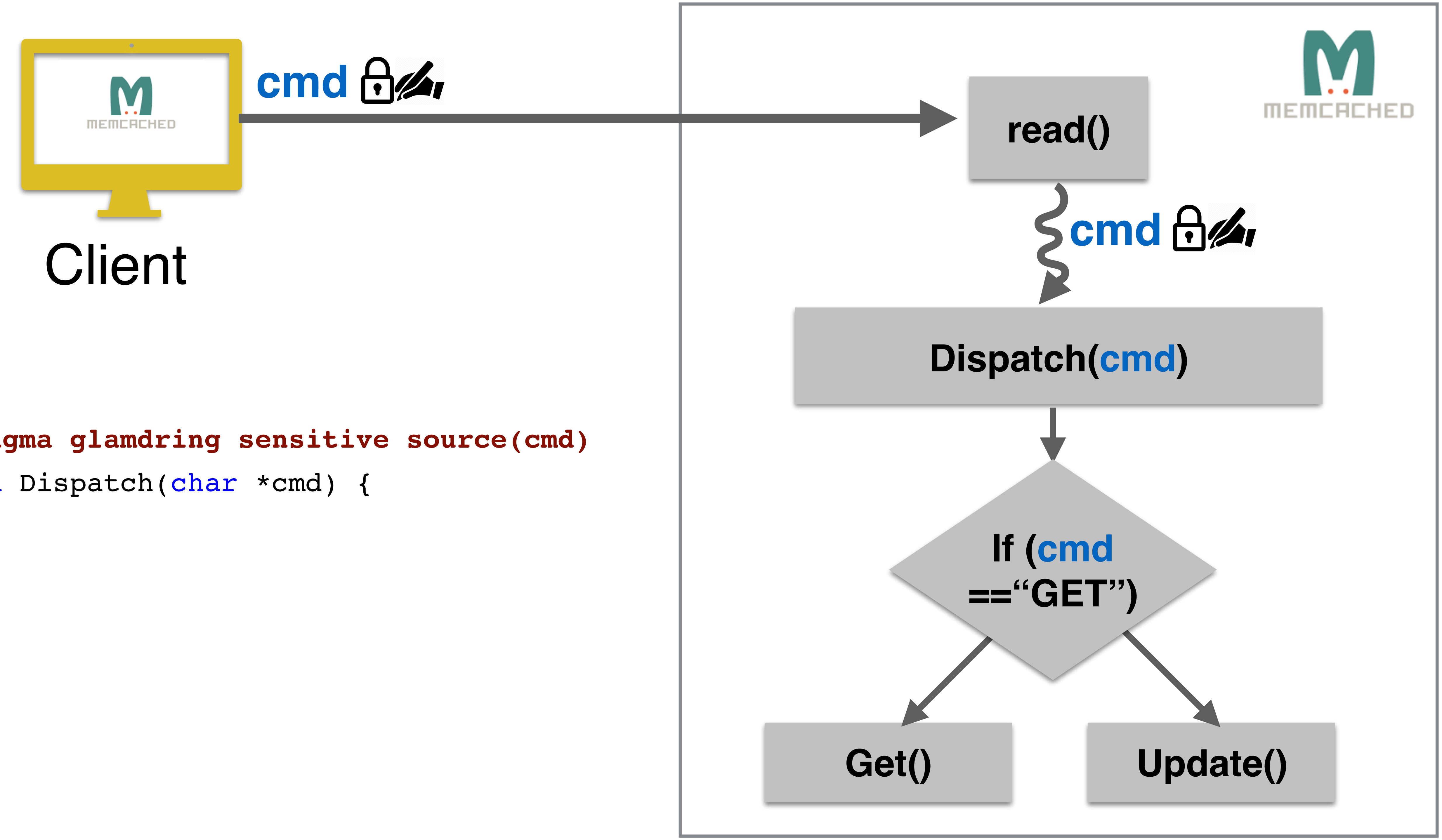




What to Annotate

- Indicate where security-sensitive data enters or leaves the program
- Sensitive data can be **encrypted** and **signed** until first use

Annotation of Security-Sensitive Data




```
#pragma glamdring sensitive source(cmd)
void Dispatch(char *cmd) {
  …
}
```

Static Analysis Goals

- Enforcing **Confidentiality**: Identify all functions that depend on sensitive data.
- Enforcing **Integrity**: Identify all functions on which the value of sensitive data depends
- Why Static Analysis?
    - Static Analysis is **conservative**, independent of the input to the program

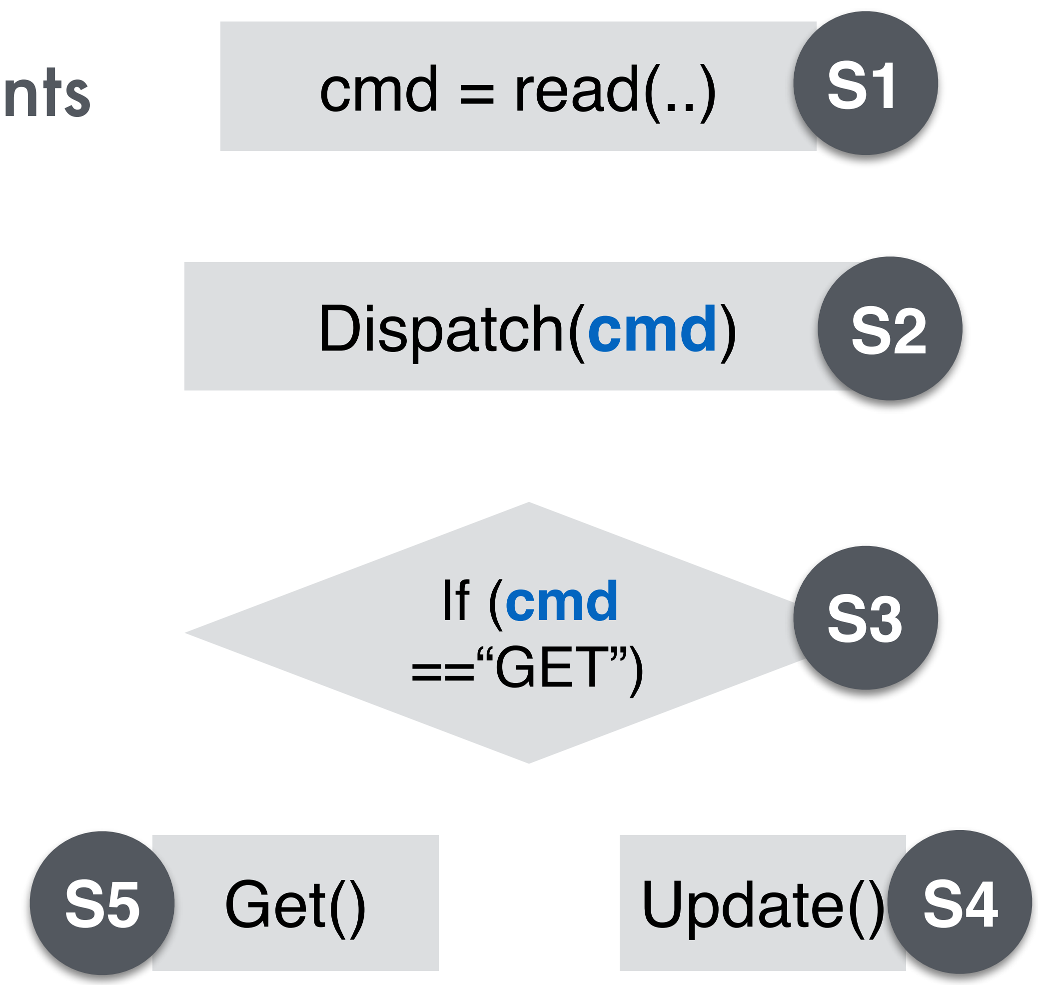
Program Dependence Graph

Captures the **control** and **data** dependencies in the program

Program Dependence Graph

Captures the **control** and **data** dependencies in the program

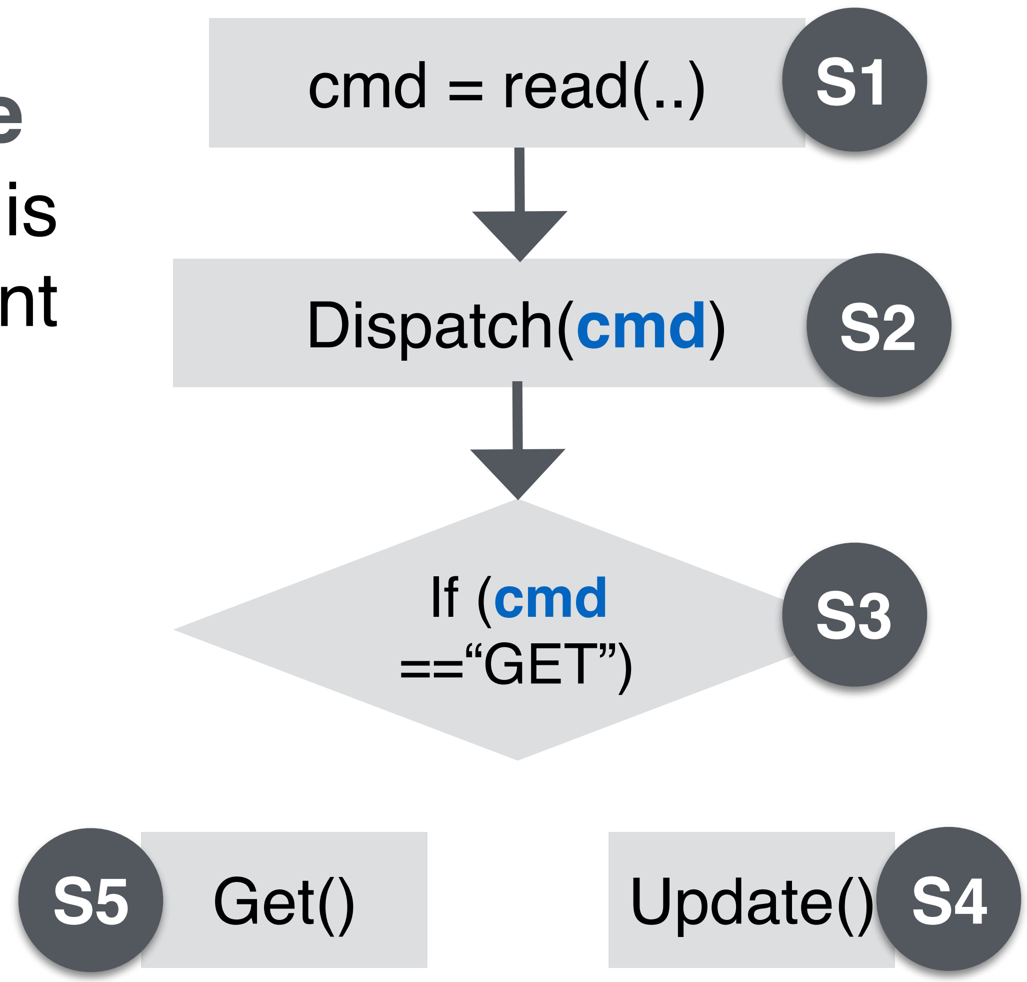
Nodes = Statements



Program Dependence Graph

Captures the **control** and **data** dependencies in the program

Data Dependence Edge
Data defined in a statement is used in the another statement

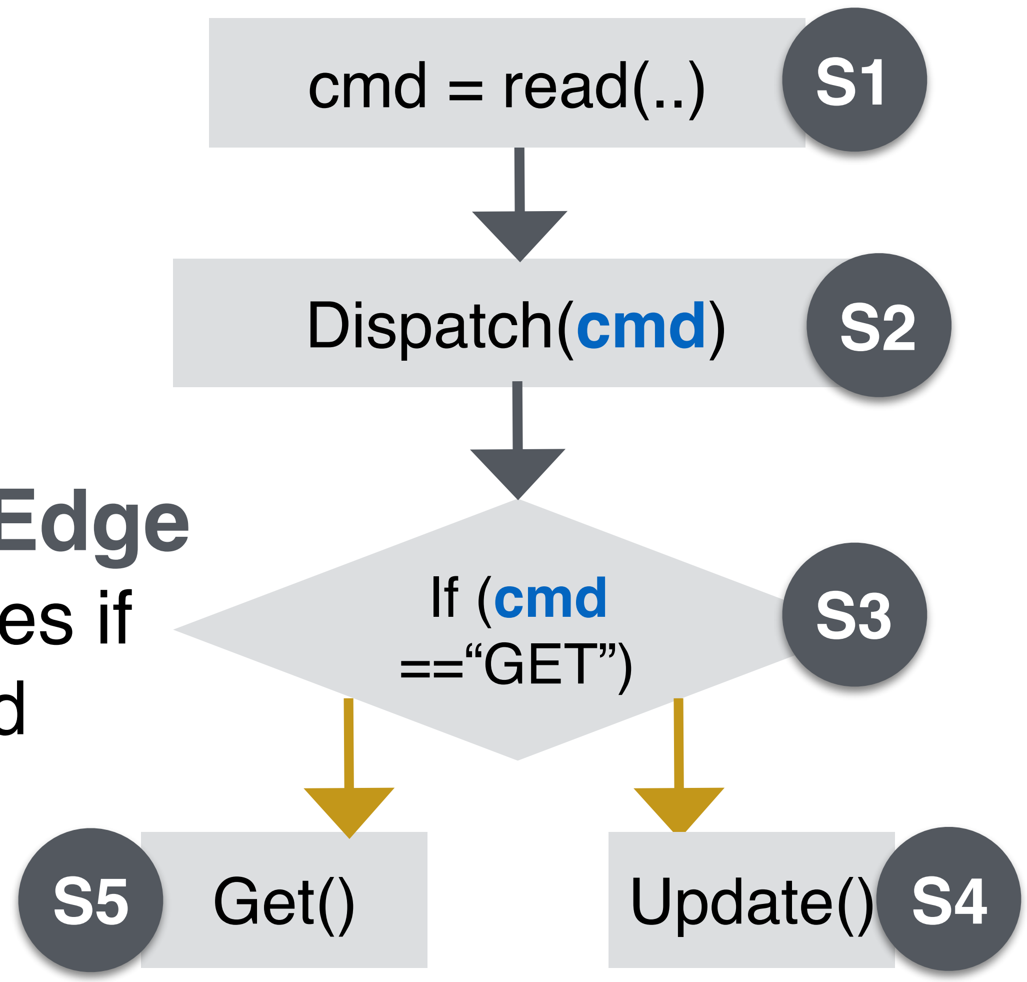


Program Dependence Graph

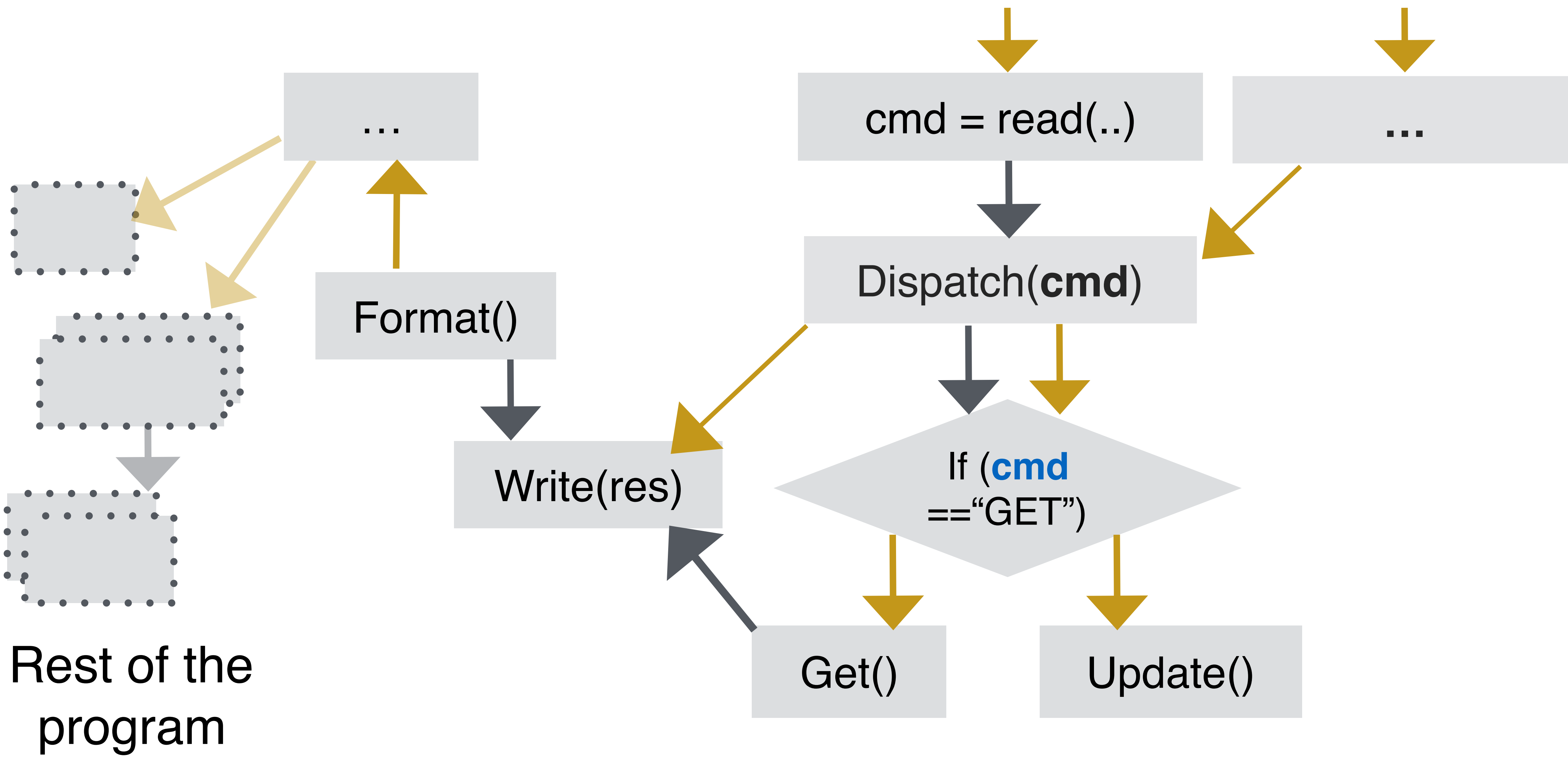
Captures the **control** and **data** dependencies in the program

Control Dependence Edge

One Statement determines if another gets executed

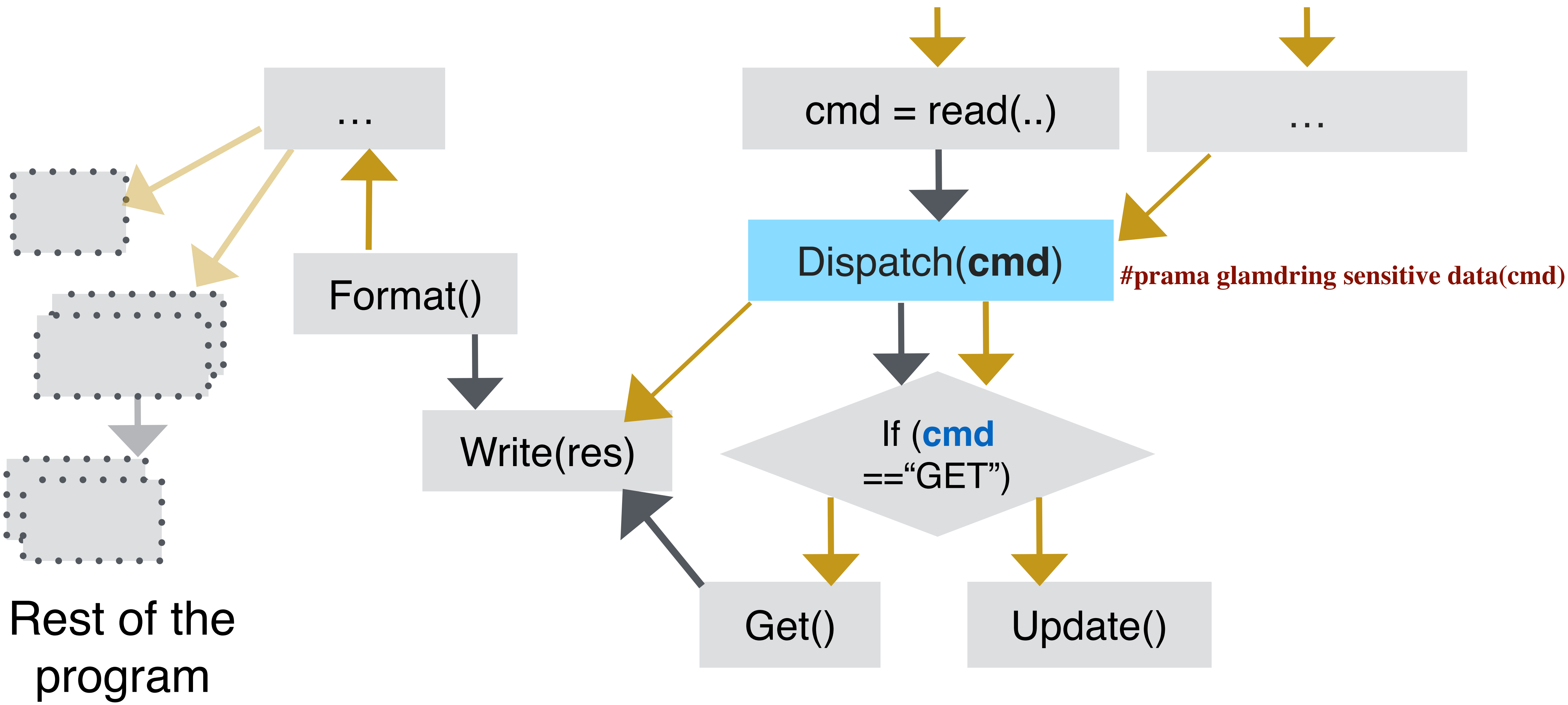


Program Dependence Graph



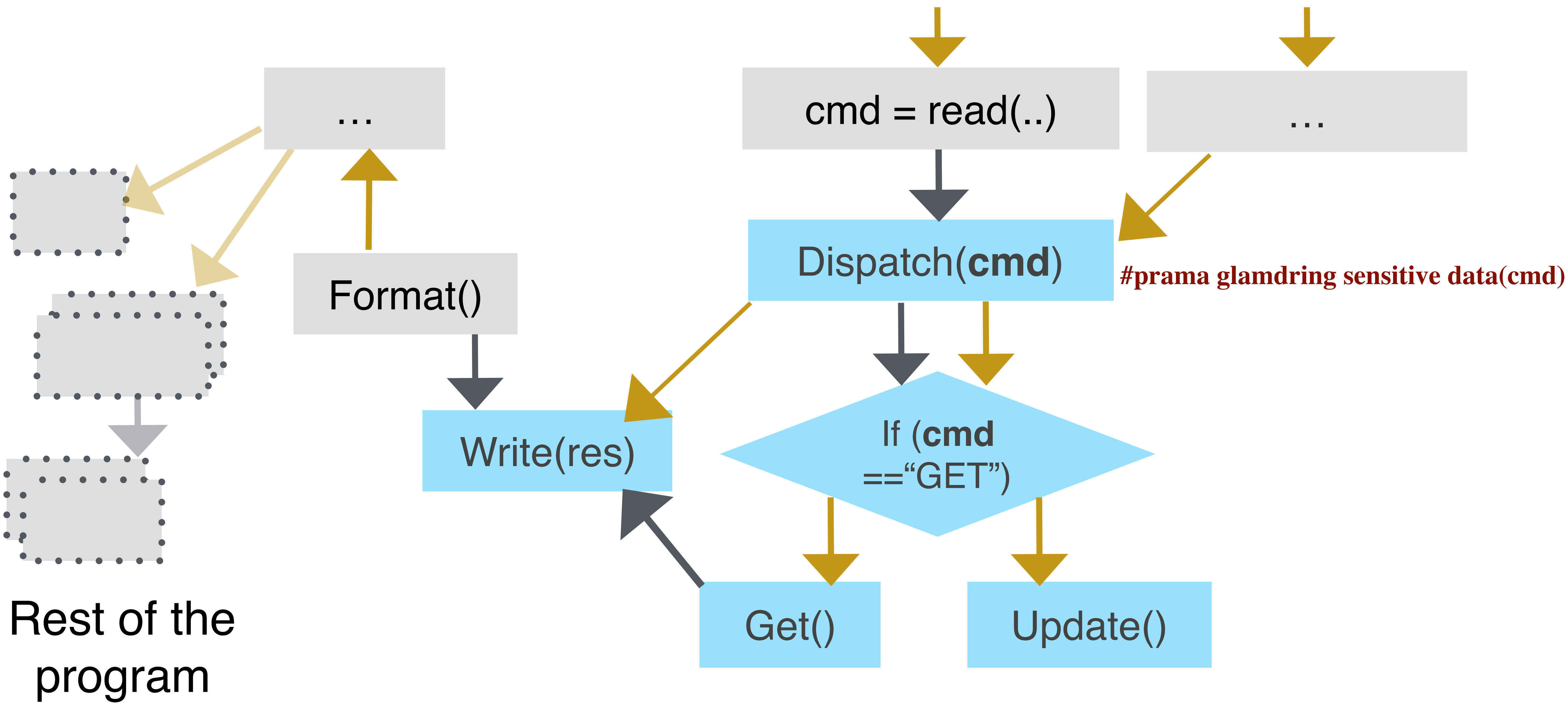
Forwards Dataflow Analysis

Confidentiality Using Graph Reachability identify all nodes with transitive control/data dependency on annotated node



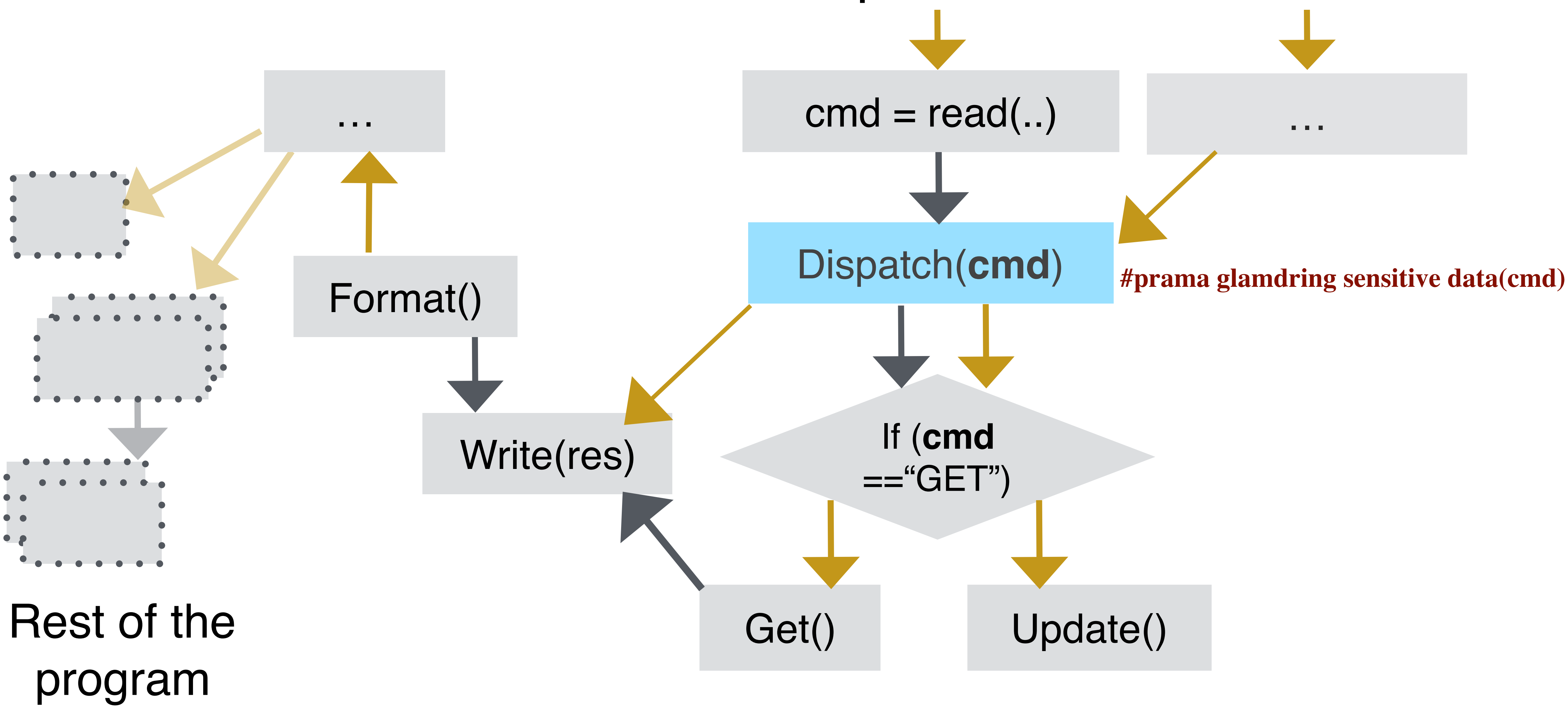
Forwards Dataflow Analysis

Confidentiality Using Graph Reachability identify all nodes with transitive control/data dependency on annotated node



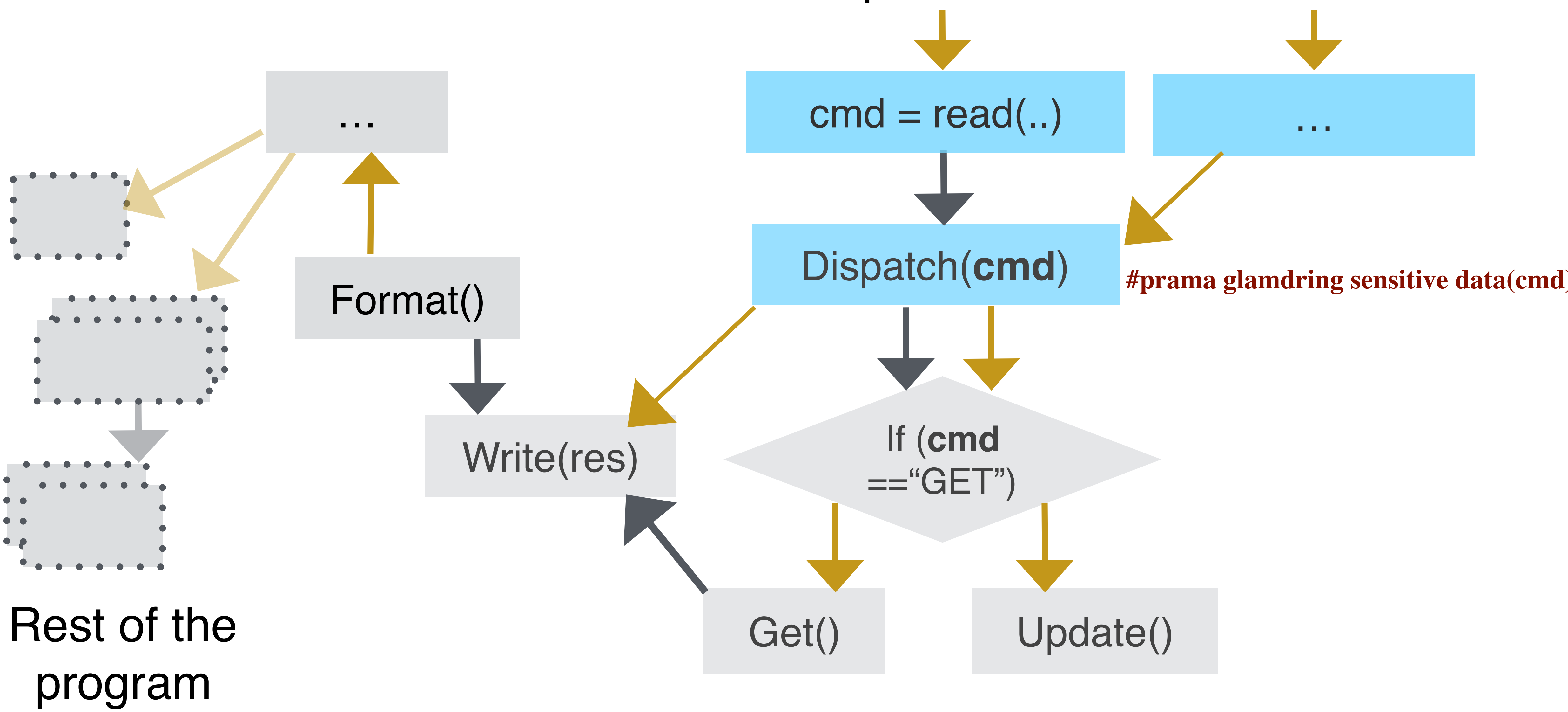
Forwards Dataflow Analysis

Integrity Using Graph Reachability identify all nodes that are transitive control/data dependent on annotated node



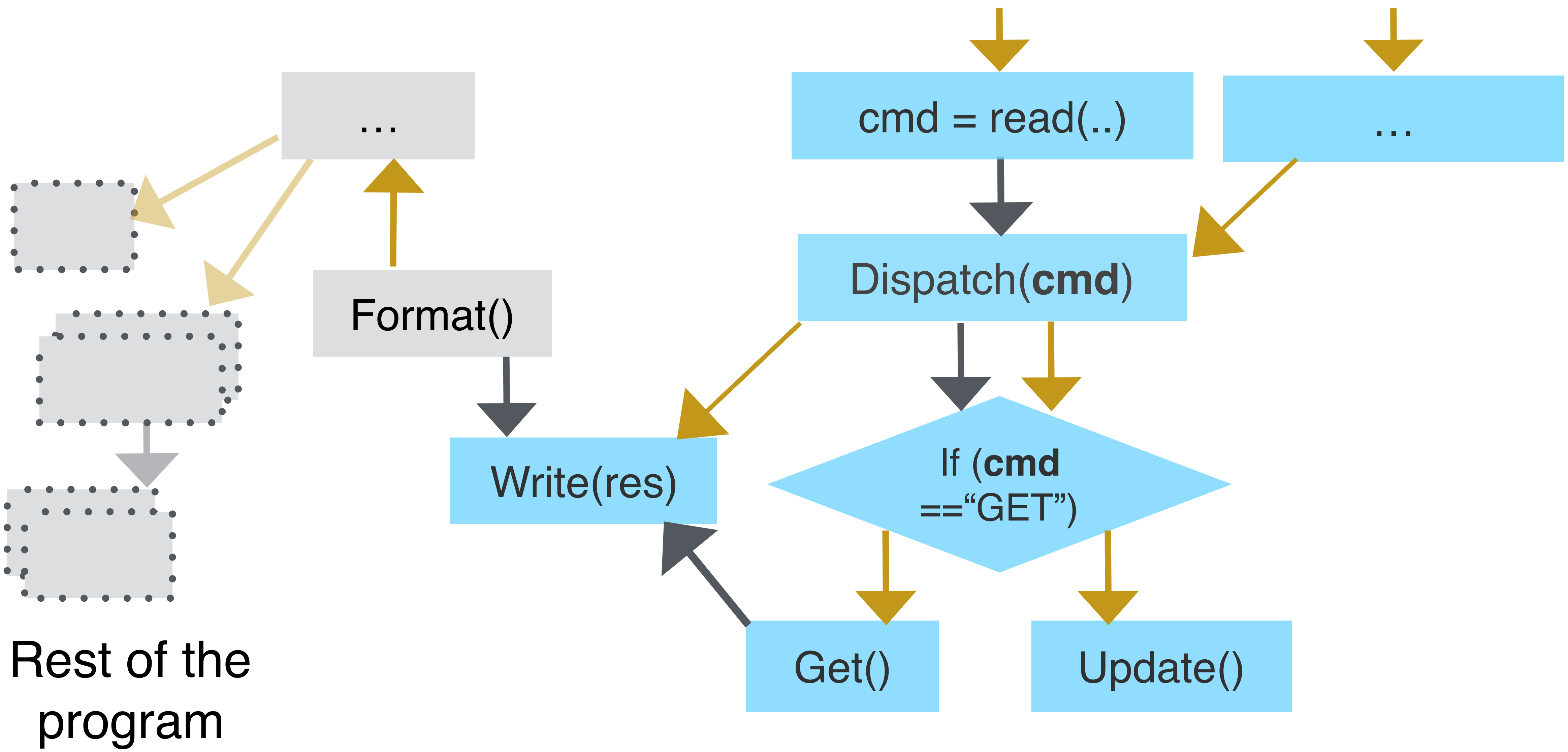
Forwards Dataflow Analysis

Integrity Using Graph Reachability identify all nodes that are transitive control/data dependent on annotated node

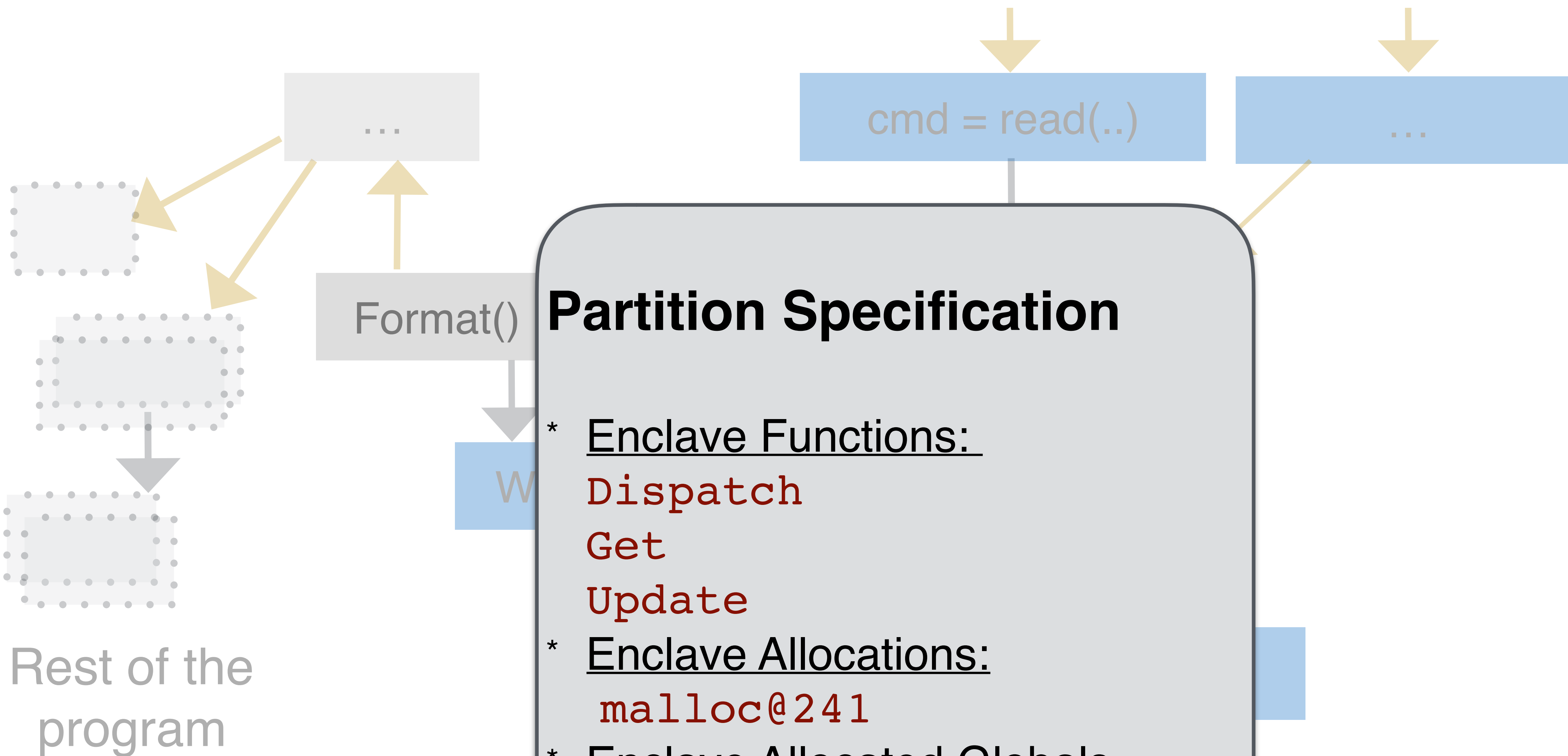


Security Sensitive Code

Union of nodes found with forwards and backwards analyses



Produce Partition Specification

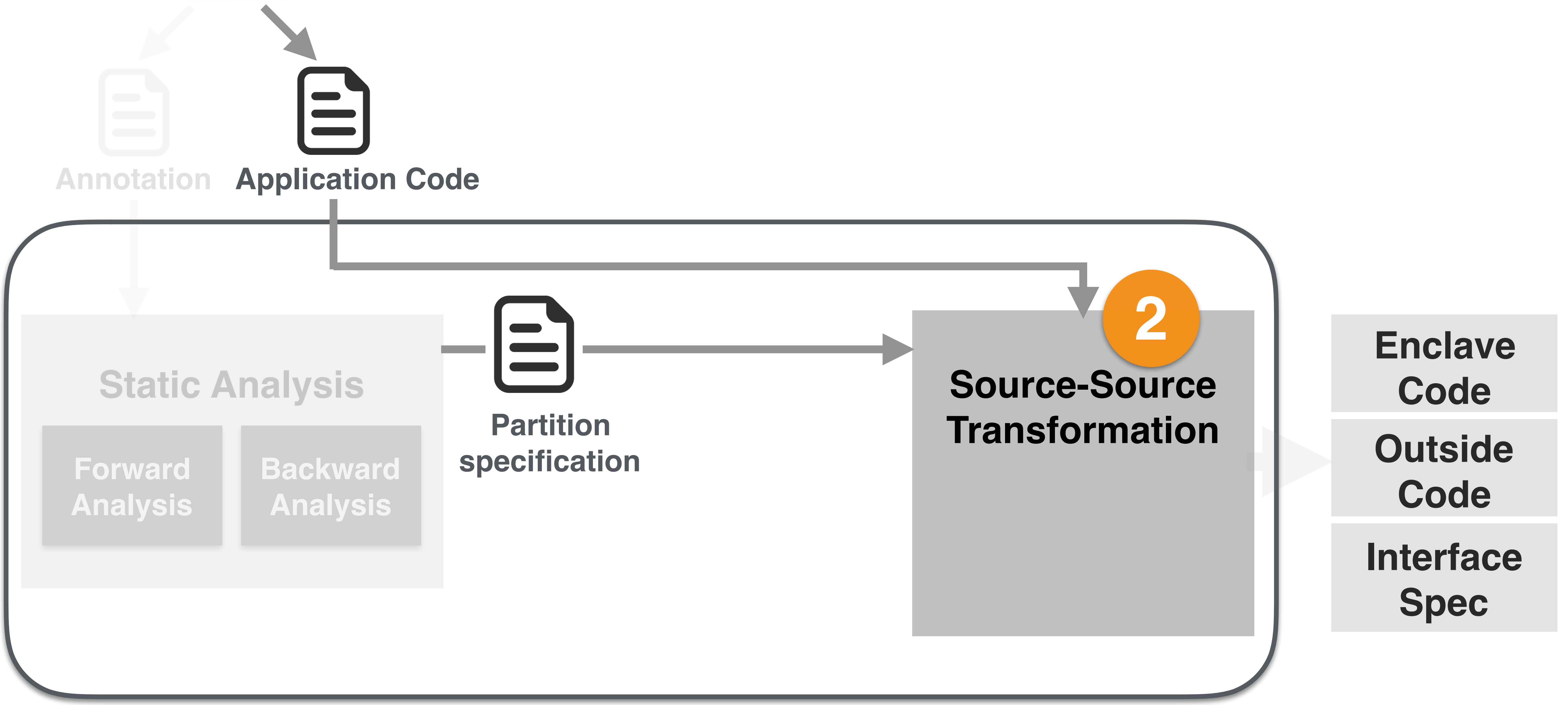


Partition Specification

* Enclave Functions:
  `Dispatch`
  `Get`
  `Update`
* Enclave Allocations:
  `malloc@241`
* Enclave Allocated Globals
  `hash_items`

2. Producing a Partitioned Application

Automatically move code into enclave and outside codebases; Generate interface specification for SDK



Source-Source Transformation

Partition Spec

* Enclave Functions:
  `Dispatch,`
  `Get,`
  `Update`
* Enclave Allocations:
  `malloc@241`
* Enclave Allocated Globals
  `hash_items`

```
void Read(…) {
   Dispatch();
}

void Dispatch(…){
…
}

void Get(…) {
…
}

void Put(…) {
…
}
```

Source-Source Transformation

Partition Spec

- Enclave Functions:
  `Dispatch,`
  `Get,`
  `Update`
- Enclave Allocations:
  `malloc@241`
- Enclave Allocated Globals
  `hash_items`

```
void Read(…) {
   Dispatch();
}
- - - - - - - - - - - - - - - - - -
void Dispatch(…){
…
}

void Get(…) {
…
}

void Put(…) {
…
}
```

Source-Source Transformation

Partition Spec

* <u>Enclave Functions:</u>
  `Dispatch,`
  `Get,`
  `Update`
* <u>Enclave Allocations:</u>
  `malloc@241`
* <u>Enclave Allocated Globals</u>
  `hash_items`

Outside

```
void Read(…) {
   ecall__Dispatch();
}
```

Enclave

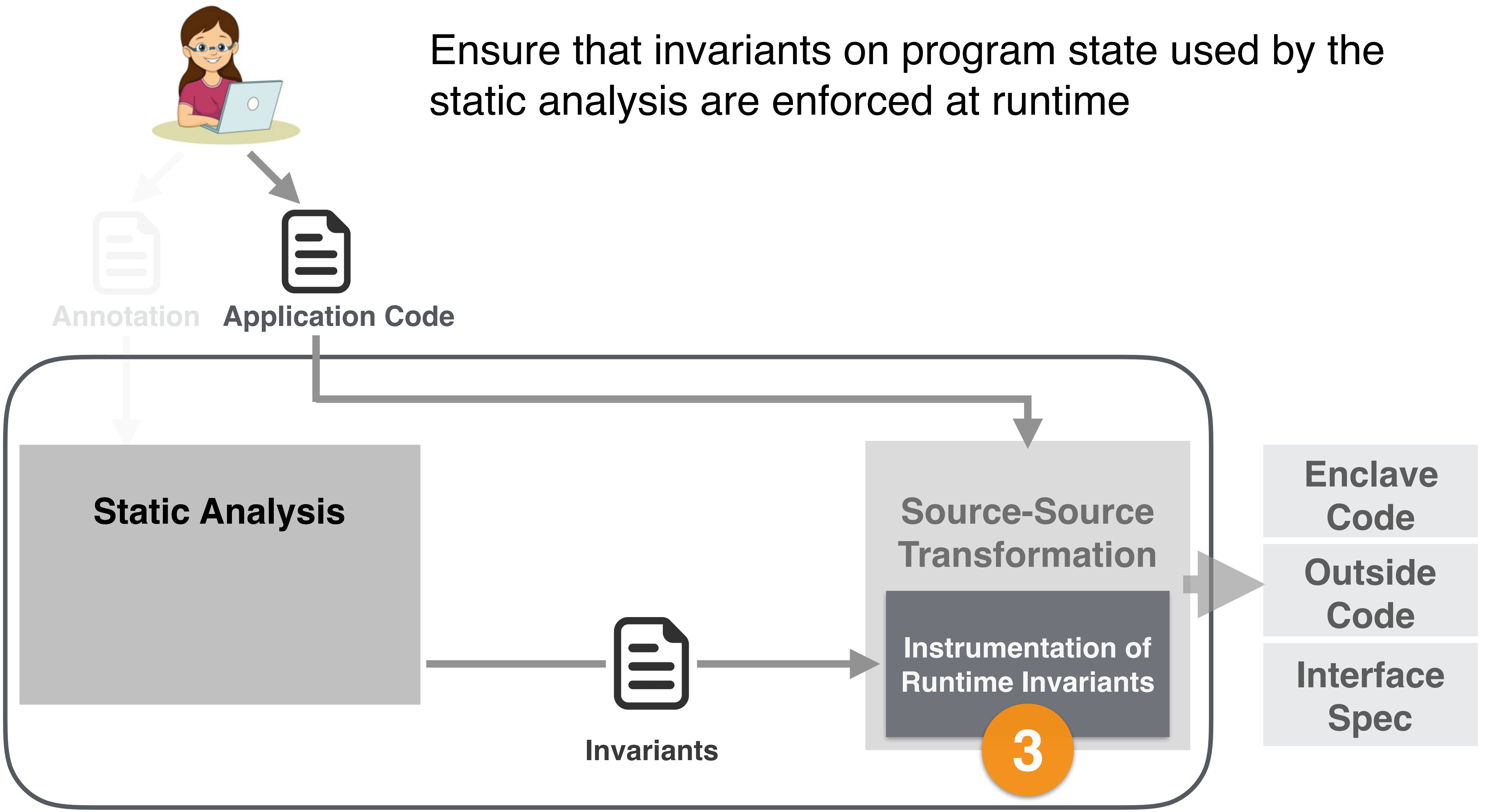
```
void ecall__Dispatch(…){
…
}

void Get(…) {
…
}

void Put(…) {
…
}
```

3. Upholding Static Analysis Invariants

Ensure that invariants on program state used by the static analysis are enforced at runtime



Infeasible Program Paths

Problem

Static Analysis prunes infeasible paths by inferring invariants on program state

```
int flag = 0;

int SomeFunc() {
   if(flag == 1)
          memcpy(data, sensitive_data);
     else
          memcpy(data, declassify(sensitive_data));
   Write(data);
}
```

Infeasible Program Paths

Problem

Static Analysis prunes infeasible paths by inferring invariants on program state

```
int flag = 0;  /* flag == 0 */

int SomeFunc() {
   if(flag == 1)
         memcpy(data, sensitive_data);
     else
         memcpy(data, declassify(sensitive_data));
   Write(data);
}
```

Violating Static Analysis Invariants

Problem

Attacker controlling untrusted code can violate the assumptions made by static analysis after partitioning

```
int flag = 0;
```

```
int SomeFunc() {

   if(flag == 1)
          memcpy(data, sensitive_data);
     else
          memcpy(data, declassify(sensitive_data));
   Write(data);
}
```

Enclave

Adding Runtime Invariant Checks

Solution

Add assertions to enforce statically inferred invariants on program state

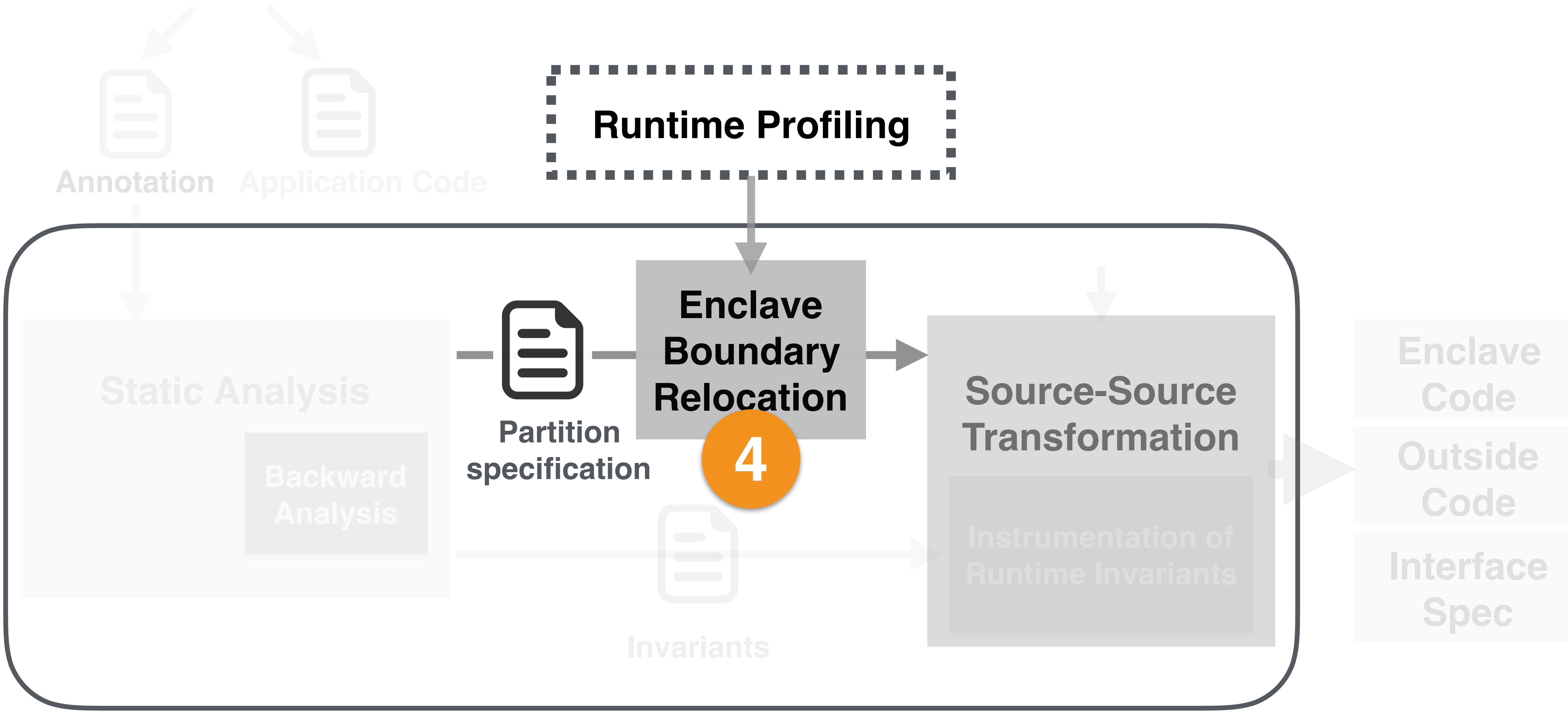
```
int flag = 0;
```

```
int SomeFunc() {
+   assert(flag == 0);
    if(flag == 1)
            memcpy(data, sensitive_data);
        else
            memcpy(data, declassify(sensitive_data));
    Write(data);
}
```

Enclave

4. Improving Performance After Partitioning

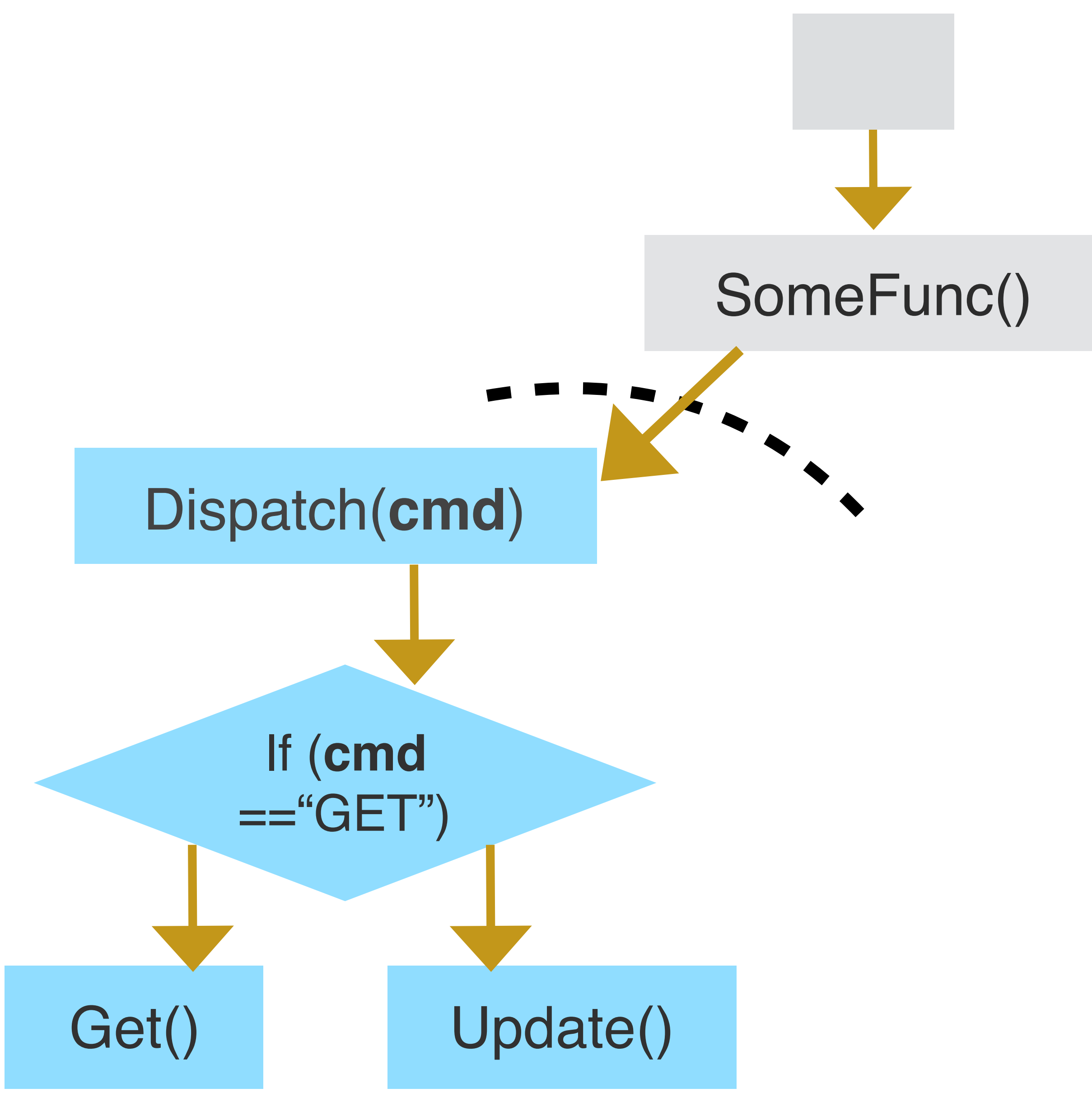
Use results of runtime profiling to remove expensive functions from enclave interface



Performance of Partitioned Applications

Expensive Interface Functions

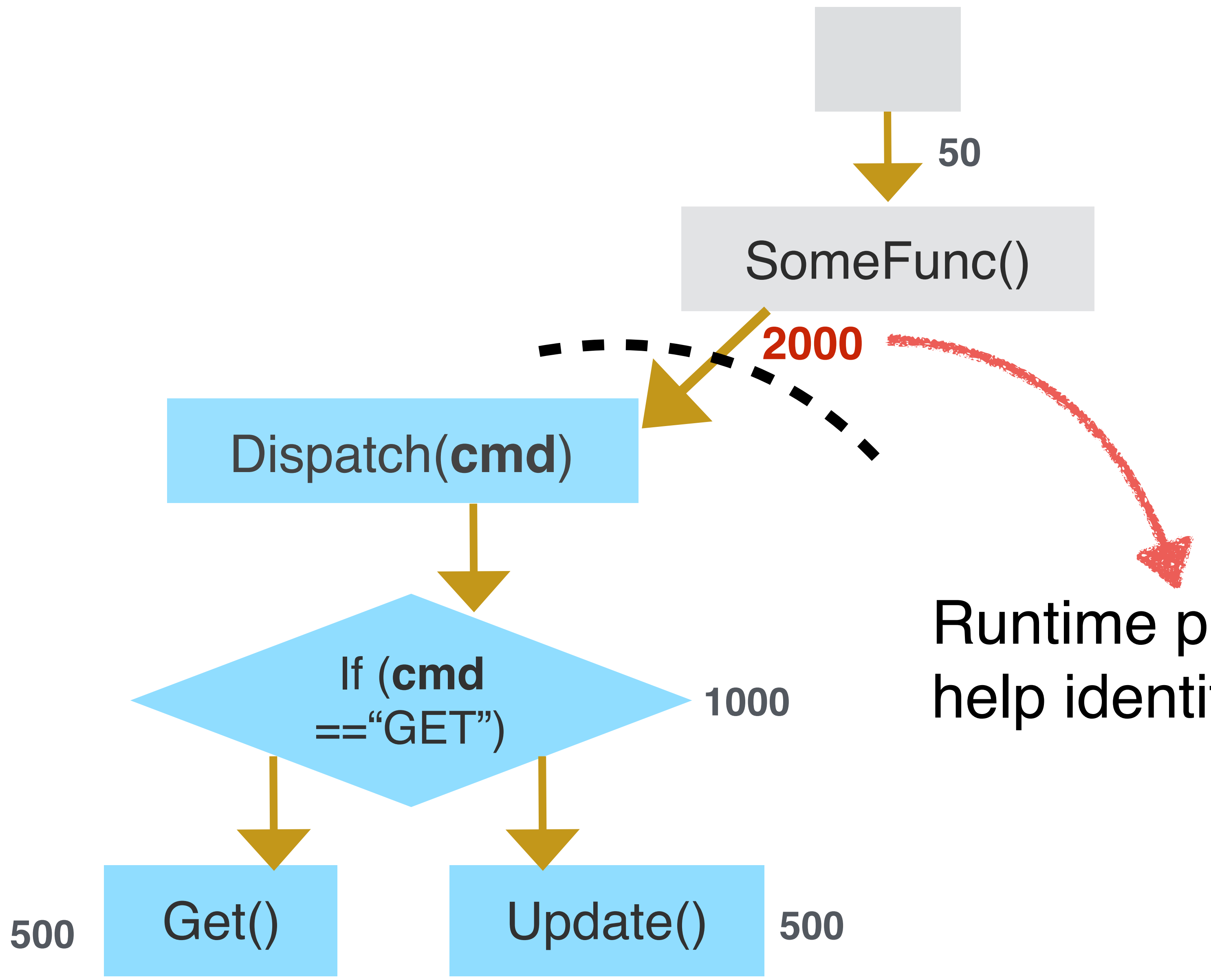
Some of the interface functions may be 'hotspots' called too frequently



Performance of Partitioned Applications

Expensive Interface Functions

Some of the interface functions may be ‘hotspots’ called too frequently

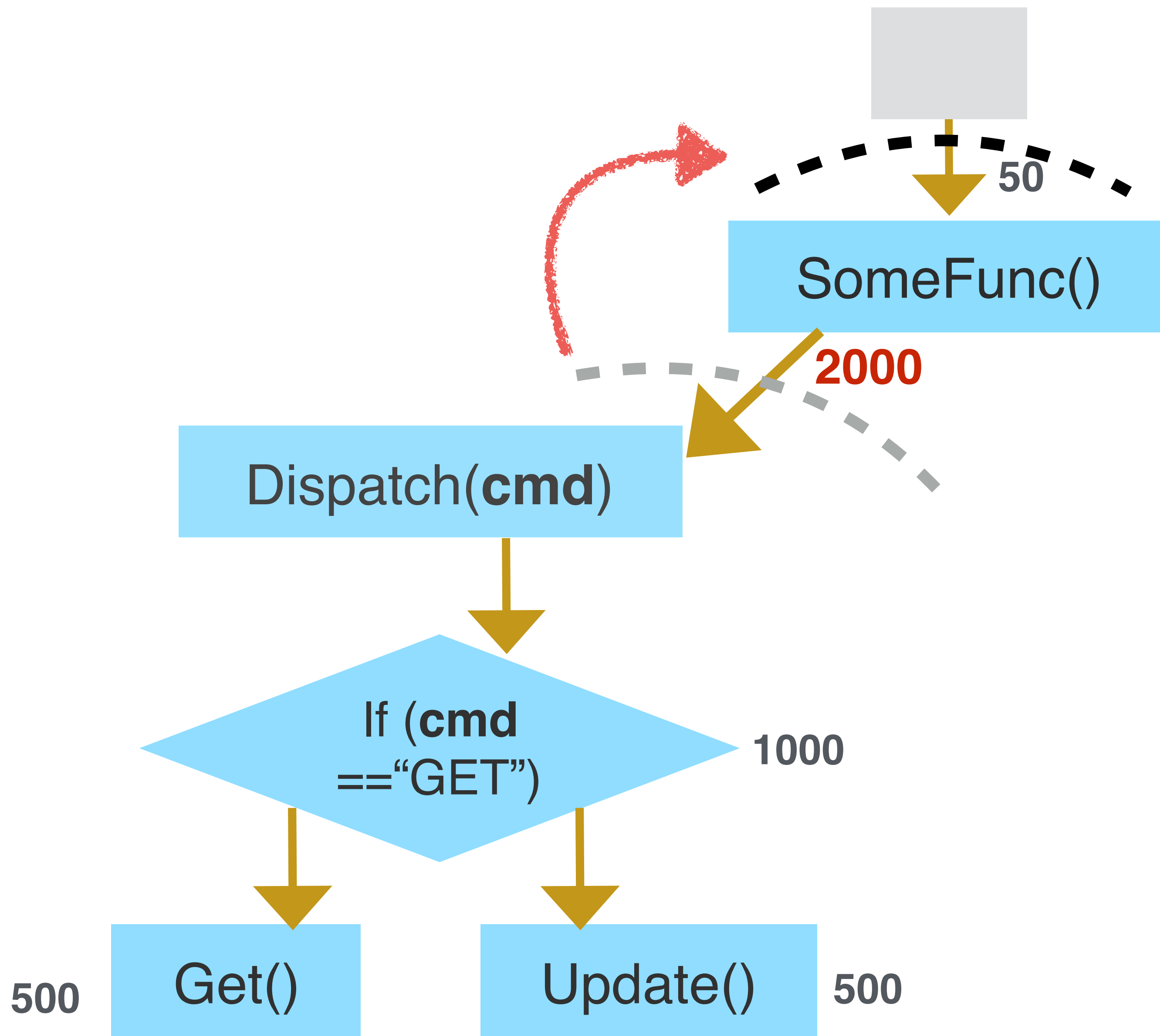


Runtime profiling can help identify hotspots

Enclave Boundary Relocation

Adding Functions to Enclave

Move additional functions into enclave to create a new interface that avoid ‘hotspots’



Evaluation Goals

- **How does Glamdring compare to other design choices**
  - Security: Size of TCB
  - Performance: Throughput

Applications and Implementation

Application	Data	Confidentiality	Integrity
Memcached	Key-Value pairs	Yes	Yes
LibreSSL	CA Root certificate	Yes	Yes
Digital Bitbox	Private Keys	Yes	Yes

Implementation

- **Static Analysis**:
  - Existing tools
- **Code Generation:**
  - LLVM/Clang 3.9 — around 5000 LoC

Security Evaluation - TCB size

How big is the TCB of applications?

Applications	Code Size (kLoC)	TCB size
Memcached	31	12 (**40%**)
DigitalBitbox	23	8 (**38%**)
LibreSSL	176	38 (**22%**)

TCB is less than 40% of the application size

Security Evaluation - TCB size

TCB size comparison with Graphene and SCONE

Applications	TCB size (kLoC)	Binary Size
Memcached (Glamdring)	**42**	770 kB
Memcached (SCONE)	149	3.3 MB
Memcached (Graphene)	746	4.1 MB

Security Evaluation - TCB size

TCB size comparison with Graphene and SCONE

Applications	TCB size (kLoC)	Binary Size
Memcached (Glamdring)	**42**	770 kB
Memcached (SCONE)	149	3.3 MB
Memcached (Graphene)	746	4.1 MB

1/3 size of TCB when using SCONE

Security Evaluation - TCB size

TCB size comparison with Graphene and SCONE

Applications	TCB size (kLoC)	Binary Size
Memcached (Glamdring)	**42**	770 kB
Memcached (SCONE)	149	3.3 MB
Memcached (Graphene)	746	4.1 MB

1/3 size of TCB when using SCONE

Order of magnitude less than with Graphene

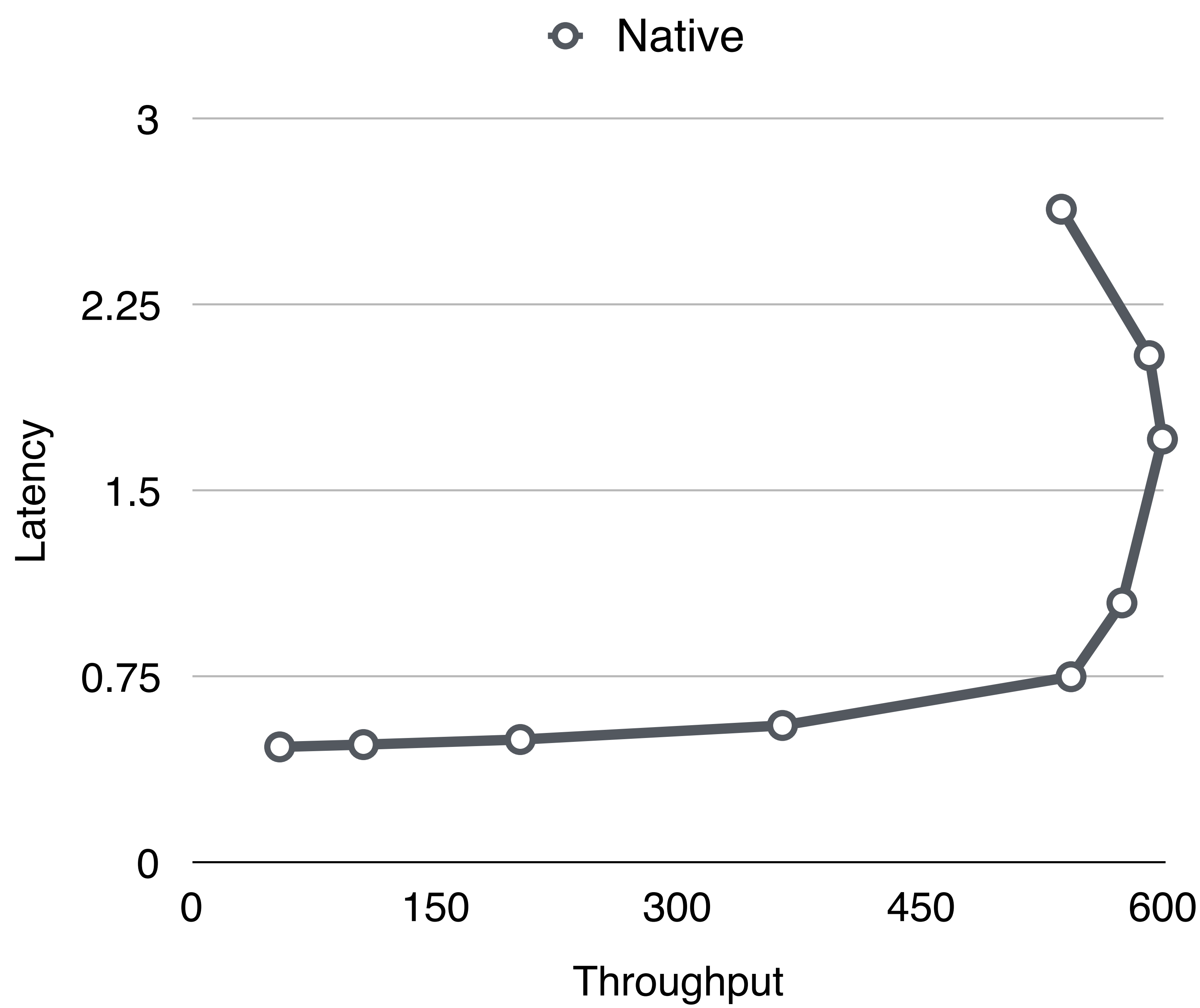
Glamdring

Comparing Performance of Design Approaches

Throughput of Memcached ported using Glamdring
with native, SCONE and Graphene

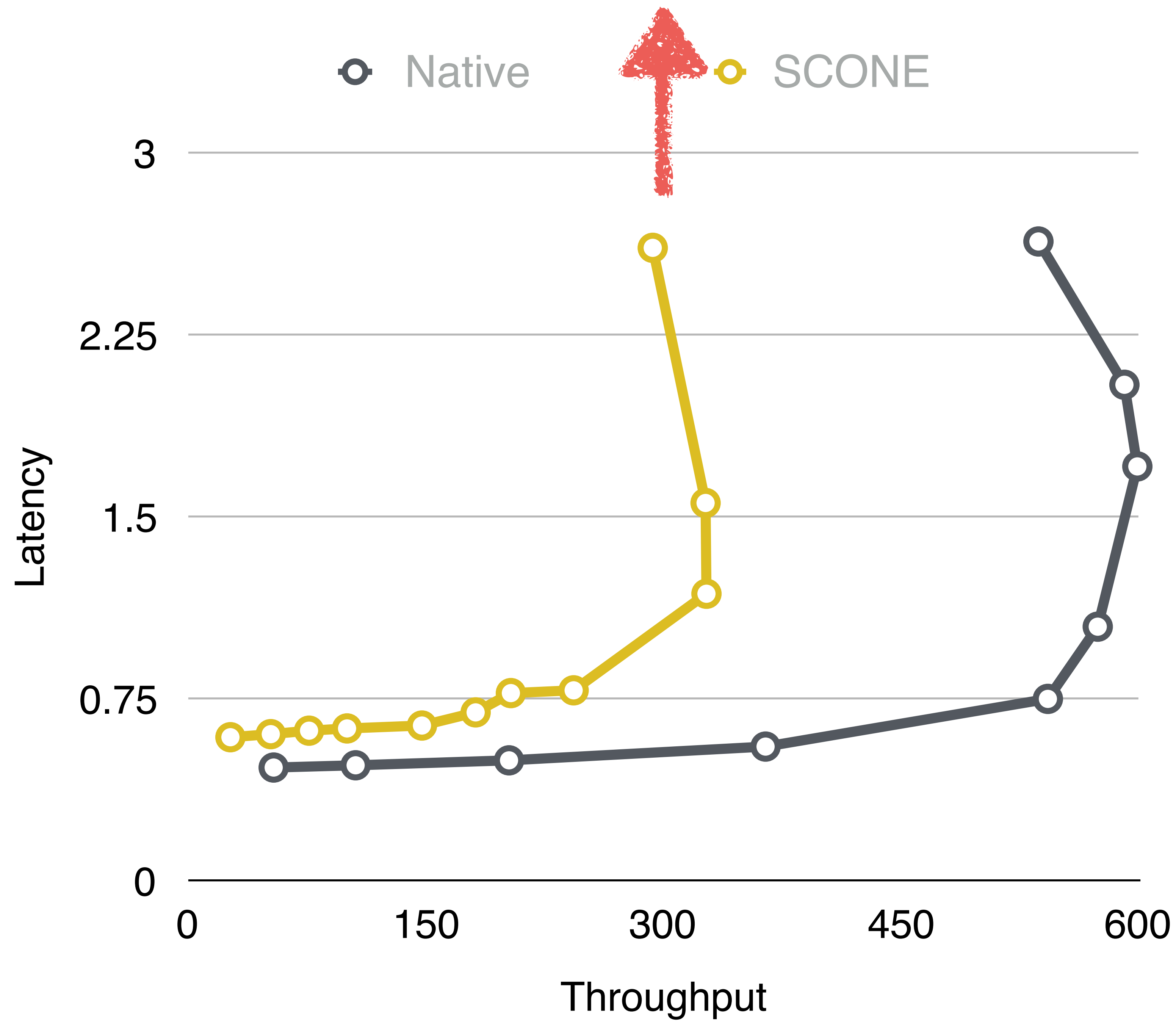
Comparing Performance of Design Approaches

Throughput of Memcached ported using Glamdring
with native, SCONE and Graphene



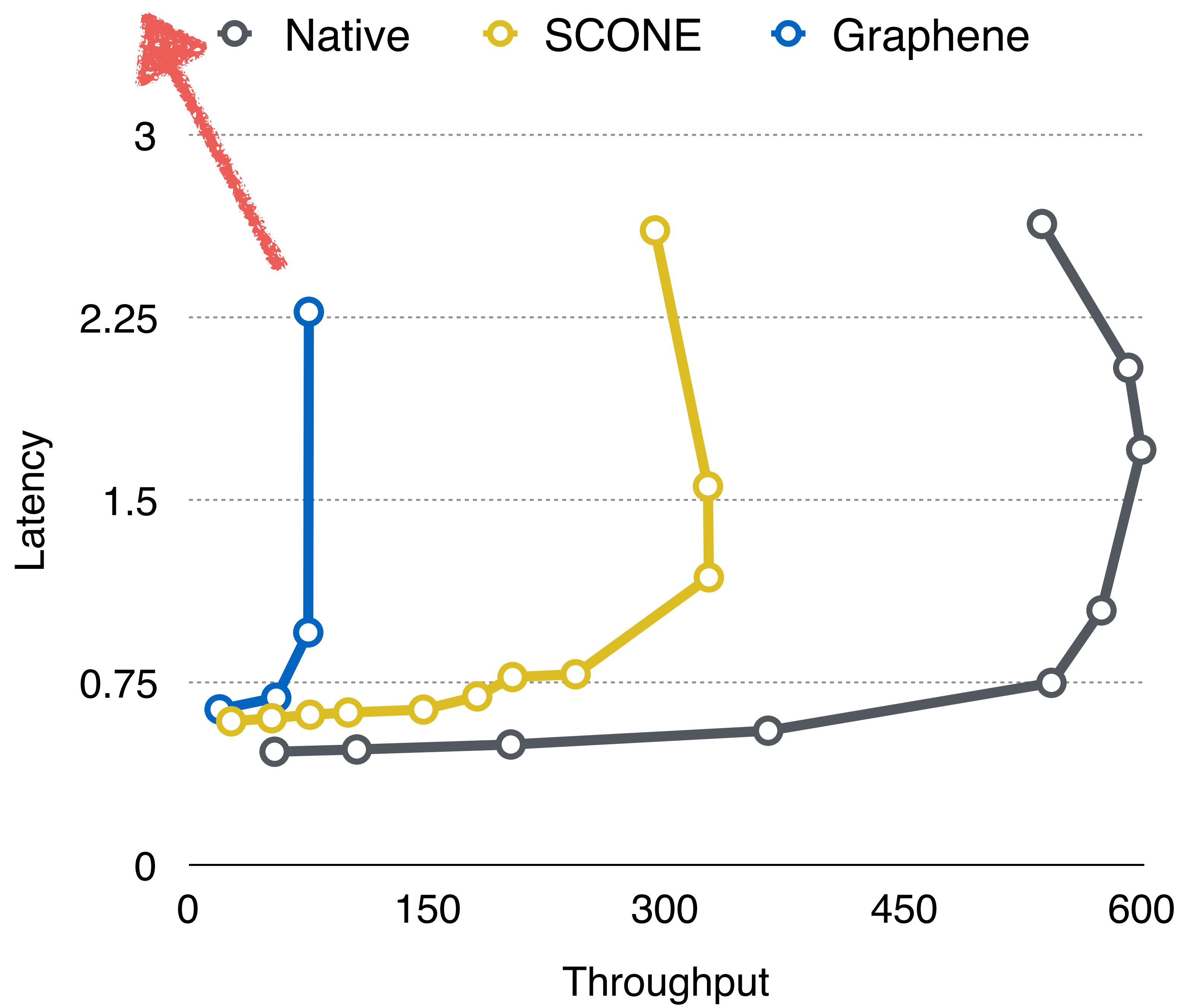
Throughput vs Latency

Avoids enclave transitions with user-level threading;
higher TCB than Glamdring

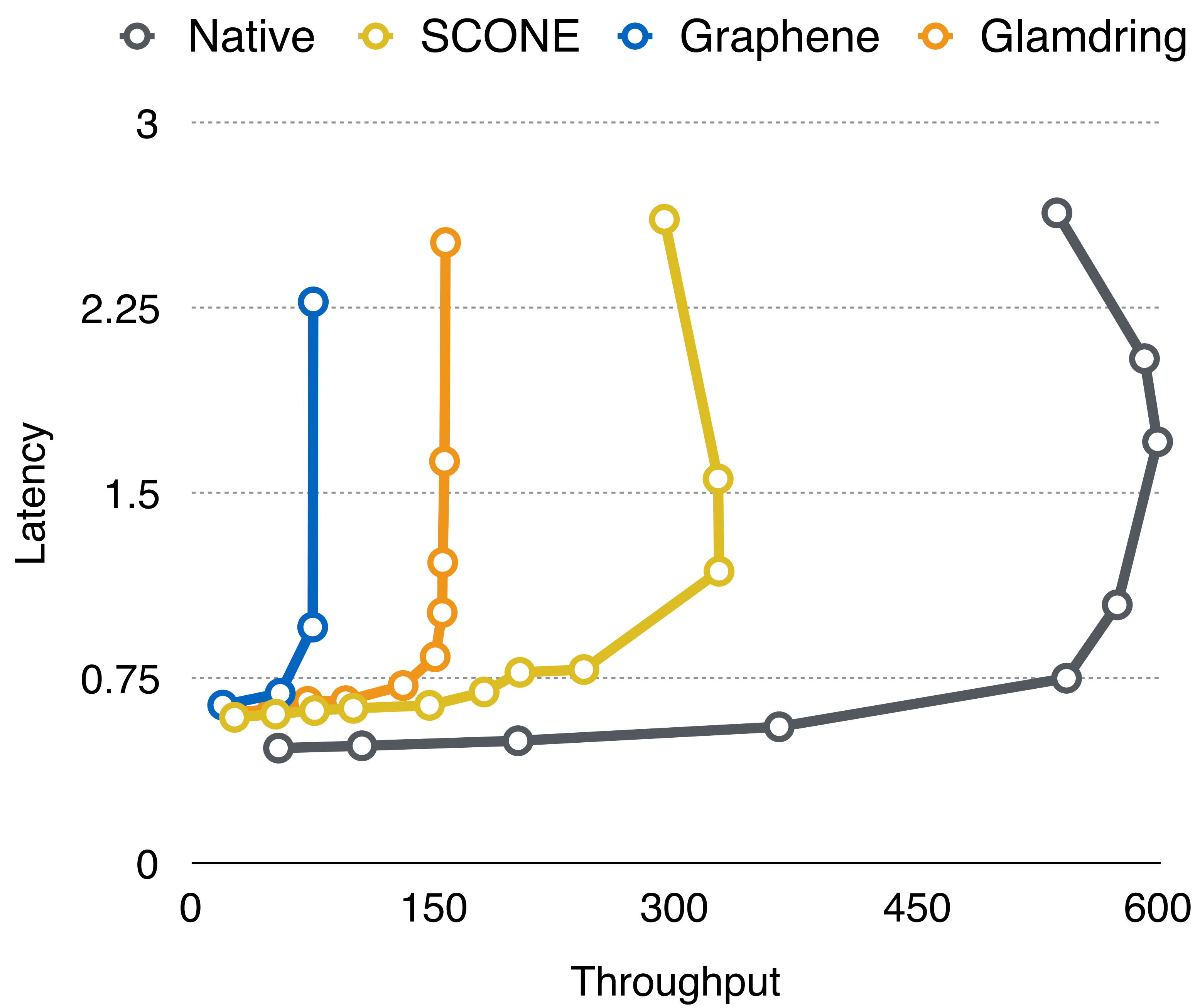


Throughput vs Latency

Entire Library OS inside enclave

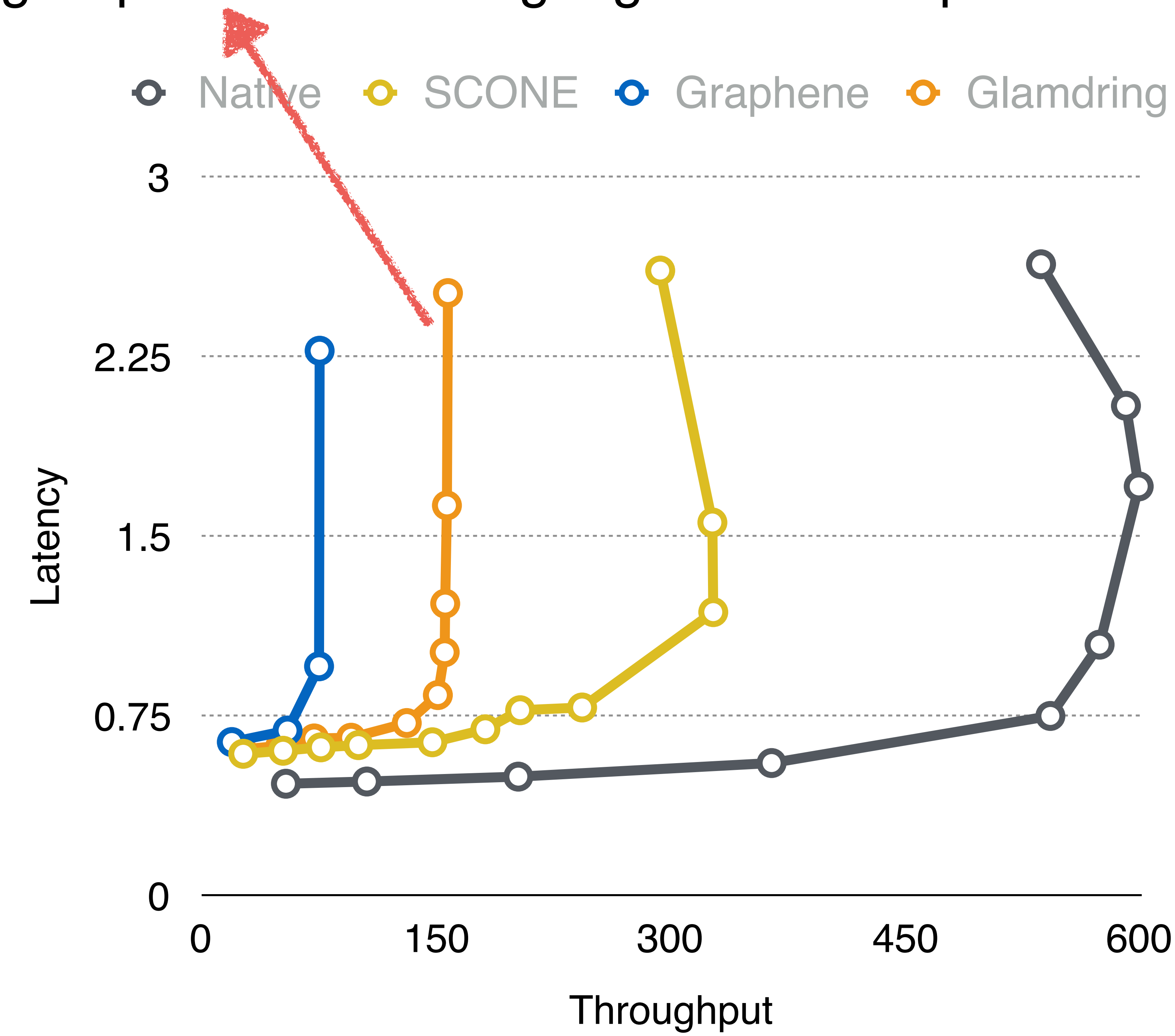


Throughput vs Latency



Throughput vs Latency

Enclave transitions dominate the cost of request handling; batching requests into multi-get gets 210k req/sec



Conclusions

- Port applications into Intel SGX enclaves with minimal TCB
- **Glamdring** — Automated program partitioning using static analysis
  - Identifies minimum TCB, produces partitioned code, enforces program state invariants, uses
- Evaluated three applications - smaller TCB than prior approaches with acceptable performance

Divya Muthukumaran

dmuthuku@imperial.ac.uk

Security Evaluation - Attacks and Defences

- **Enclave Call Ordering Attacks:** By construction. EBR does not affect this.
- **Iago Attacks:** By enforcing invariants
- **Replay Attacks:** Freshness counter
- **Enclave Code Vulnerabilities:** TCB is reduced — enables code analysis

Evaluation - Impact of EBR

How many functions were moved into the enclave, and what was the impact on enclave crossings

Application	EBR Enclave Functions	Enclave Crossings (No EBR)	Enclave Crossings (With EBR)
Memcached	1	54	6
LibreSSL	2	24,780	6727
Digital Bitbox	4	10,943	38

Glamdring

Evaluation - Impact of EBR

Even **few functions** inside….

reduced enclave crossings by orders of magnitude

Application	EBR Enclave Functions	Enclave Crossings (No EBR)	Enclave Crossings (With EBR)
Memcached	1	54	6
LibreSSL	2	24,780	6727
Digital Bitbox	4	10,943	38